

THE DATA GENERAL
NOVA 800 MINICOMPUTER
AS A DIGITAL CONTROLLER

John William Pounds

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

THE DATA GENERAL
NOVA 800 MINICOMPUTER
AS A DIGITAL CONTROLLER

by

John William Pounds, Jr.

September 1975

Thesis Advisor:

Donald E. Kirk

Approved for public release; distribution unlimited.

T169787

20.

A system operation tutorial is offered which includes a programmed instruction section to efficiently familiarize the prospective user. Finally, examples of system use as a digital controller are provided.

The Data General Nova 800 Minicomputer
As a Digital Controller

by

John William Pounds, Jr.
Lieutenant, United States Navy
B.S.E.E., University of New Mexico, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the
NAVAL POSTGRADUATE SCHOOL
September 1975

Thesis
P17529
C.1

ABSTRACT

The Data General Nova 800 minicomputer is described in detail. An analysis is made of its instruction set and a description is given of available peripheral equipment. The specification criteria for A/D and D/A devices is discussed and the selection of specific devices is supported. The DT 1620 data acquisition system and the DT 212 D/A converter are described. The interface considerations for adapting the A/D and D/A converters to the Nova system are discussed. Programming examples for the combined system are given. A system operation tutorial is offered which includes a programmed instruction section to efficiently familiarize the prospective user. Finally, examples of system use as a digital controller are provided.

TABLE OF CONTENTS

I.	INTRODUCTION-----	10
	A. PROBLEM STATEMENT-----	11
II.	THE NOVA 800 COMPUTER-----	13
	A. INTRODUCTION-----	13
	B. THE INSTRUCTION SET-----	15
	1. Memory Reference Instructions-----	16
	2. Arithmetic and Logical Instructions-----	18
	3. I/O Instructions-----	21
	4. Mnemonics-----	23
	C. THE DATA CHANNEL-----	24
	D. SPECIAL OPTIONS-----	25
	E. THE OPERATING SYSTEM-----	26
	1. Self-Loading Bootstrap and Image Core Loader/Writer-----	26
	2. Command Line Interpreter-----	28
	3. Text Editor-----	31
	4. Assembler-----	34
	5. Relocatable Loader-----	38
	F. GENERAL PROGRAMMING-----	41
III.	DATA ACQUISITION AND DISTRIBUTION-----	47
	A. INTRODUCTION-----	47
	B. RESOLUTION AND SPEED COSTS-----	47
	C. THE INTERNAL ANALOG PERIPHERAL-----	49

D.	GENERAL INTERFACE EQUIPMENT -----	51
E.	THE DT 1620 -----	56
1.	Multiplexer and Differential Amplifier -----	58
2.	Sample and Hold Circuit-----	61
3.	12-bit A/D Converter -----	61
4.	Special Options-----	62
F.	THE DT 212 -----	63
G.	INTERFACE CONSIDERATIONS-----	64
1.	Device Selection -----	64
2.	Control Signals -----	65
3.	Setting the DONE Flipflop -----	68
H.	CALIBRATION OF THE A/D AND D/A CONVERTERS -----	70
I.	SYSTEM PROGRAMMING -----	71
IV.	SYSTEM OPERATION TUTORIAL-----	79
A.	INTRODUCTION -----	79
B.	MACHINE OPERATION -----	79
C.	SYSTEM FAMILIARIZATION -----	83
V.	EXAMPLE CONTROL PROBLEMS -----	88
A.	INTRODUCTION -----	88
B.	PROGRAM DEVELOPMENT -----	89
C.	MINIMUM PROTOTYPE DESIGN -----	95
D.	RIPPLE FREE RESPONSE -----	97

VI.	CONCLUSION	102
A.	EVALUATION OF THE NOVA COMPUTER	102
B.	EVALUATION OF THE A/D EQUIPMENT	103
C.	EVALUATION OF THE D/A EQUIPMENT	103
D.	EVALUATION OF THE SYSTEM	103
E.	RECOMMENDATIONS FOR EXPANSION.....	104
APPENDIX A	Instruction Mnemonics	105
APPENDIX B	Basic A/D and D/A Conversion	116
B.1	Basic D/A Conversion	118
B.2	Successive Approximation A/D Conversion System	120
B.3	Counter Ramp A/D Conversion	120
B.4	Parallel A/D Conversion.....	122
APPENDIX C	Bus Signals	124
APPENDIX D	Device Connection and Specification	129
D.1	Device Connection	129
D.2	DT 1620 Functional Pin Description	132
APPENDIX E	Assembler Error Flags.....	142
APPENDIX F	Manufacturer Supplied Subroutines	144
APPENDIX G	Instruction Execution Times.....	152
BIBLIOGRAPHY	154
INITIAL DISTRIBUTION LIST	155

LIST OF FIGURES

Figure

1.1	Basic Interface Elements-----	10
2.1	Memory Reference Instruction Formats-----	16
2.2	Arithmetic and Logic Instruction Formats -----	18
2.3	Organization of Arithmetic Unit -----	20
2.4	I/O Instruction Format -----	21
3.1	Relative Cost vs Resolution -----	48
3.2	Data Bus Control Circuit -----	52
3.3	Control Signal Inverters -----	53
3.4	Device Select Circuit -----	54
3.5	Done and Busy Flipflop Control -----	55
3.6	DT 1620 Simplified Schematic-----	57
3.7	Common Mode Voltage Model-----	58
3.8	Single-Ended Input Mux -----	59
3.9	Differential Input Mux-----	60
3.10	DT 1620 Output Coding -----	62
3.11	DT 212 Simplified Schematic-----	63
3.12	Device Select Separation -----	65
3.13	Relative Timing Diagram for Programmed Transfers -----	68
5.1	Example Control System -----	88
5.2	Comparison of Continuous and Sampled Systems -----	99

5.3	Minimum Prototype Response -----	100
5.4	Minimum Ripple Response-----	101
B-1	Example Digital Coding -----	116
B-2	Digital Word vs Equivalent Analog Voltage -----	117
B-3	Basic D/A Conversion -----	119
B-4	Successive Approximation A/D Converter -----	121
B-5	Counter Ramp A/D Converter-----	122
B-6	Parallel A/D Conversion -----	123

I. INTRODUCTION

During the past decade, the use of the digital computer in complex control systems has increased at an extraordinary rate. System designs incorporating the use of a digital machine have become commonplace in industry as well as the military. These sample-data systems are being designed and interfaced to existing systems to improve reliability, speed of response, and accuracy. Even though the individual systems differ in complexity the basic configurations are the same.

The basic elements of any interface between a digital processor and an analog system are shown in Figure 1.1.

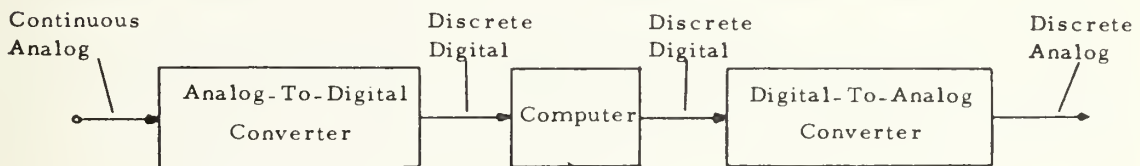


Figure 1.1 Basic Interface Elements

The analog-to-digital converter samples the continuous signal on command by the computer. It then converts the voltage sampled into a digital word of the form used by the computer. The computer performs some specified operations on the word and feeds the resulting digital output to the digital-to-analog converter. The digital-to-analog converter transforms the digital word into a discrete analog pulse.

One point of contention between system designers has been whether to design a special purpose computer for a given task or to use a general purpose computer which may be used in a variety of tasks. This decision is dependent upon the problem to be solved. For the special purpose computer, the advantages are smaller size, less cost and ease of interface design. For the general purpose computer, the major advantage is flexibility through multiprocessing.

In recent years, the size of general purpose computers has diminished with the emergence of the minicomputer. Large core memories have given way to the use of electronic memories using Large Scale Integration (LSI). And the large magnetic tape reel-to-reel units have been replaced by cassette tape storage. The minicomputer and its associated mini-peripherals are in general slower, less powerful, and less sophisticated than the large general purpose machines, but have the advantages of smaller size, lower cost, and simplicity while still having a great deal of flexibility. Recently, through additional miniaturization, the interface elements have become small enough so that they may be placed inside a minicomputer chassis on an interface board. This thesis is concerned with the incorporation of such an ADC-DAC system with the Data General Nova 800 computer.

A. PROBLEM STATEMENT

In 1974, the Electrical Engineering Department of the Naval Postgraduate School purchased a Data General Nova 800 computer for use in

the area of automatic control. The computer was not equipped with A/D or D/A devices but it was felt that an interface could be designed and built to provide these functions and thereby make it possible to use the system as a digital controller. The problem was not only to interface the hardware but also to provide a basic manual for machine operation, give a description of the operational computer system, and supplement the references in the area of programming.

Chapter 2 describes the Nova computer and the associated peripherals. It includes a complete analysis of the instruction set, describes the software of the operating system, and introduces the general elements of programming. The computer description presupposes a familiarity with general minicomputer organization and operation. Chapter 3 provides the specification criteria for the A/D and D/A converters, supports the selection of specific devices, describes the interface board available with the computer, gives the interface design considerations, and introduces software programming with reference to A/D and D/A devices. Chapter 4 is a system tutorial. It provides operating information in a concise format for quick reference and a step-by-step procedure for learning the elements of computer operation and programming. Chapter 5 provides examples of the use of the system as a digital controller for a simulated continuous plant. Chapter 6 gives an evaluation of the system elements and of the system as a whole and recommends directions for expansion.

II. THE NOVA 800 COMPUTER

A. INTRODUCTION

The Nova 800 is a general purpose, 16-bit word length computer. It is organized around four accumulators, AC0 - AC3. Both AC2 and AC3 can also be used as index registers. The mainframe is $10\frac{1}{2}$ inches high and mounts inside a standard 19-inch chassis. It has 17 slots; two are used for the central processor leaving 15 available for memory and I/O interfaces. The full memory cycle time, that is the maximum time required for the processor to access one memory location, is 800 nanoseconds. All arithmetic and logical instructions are executed in a single cycle time.

The central processor provides control for the entire system. It performs all the arithmetic, logical and data handling operations, and sequences the program. The processor handles words which are stored in a memory with a capacity of 8K. The processor can interpret a word as an address, a logical word, a pair of 8-bit bytes, or a 16-bit signed or unsigned binary number. The arithmetic instructions operate on fixed point signed or unsigned binary numbers using 2's complement conventions.

The processor sequences through a program by executing instructions retrieved from consecutive memory locations as pointed to by the 15-bit program counter (PC). At the end of each instruction, the PC is

incremented by one. Sequential program flow is altered by changing the contents of PC, either by replacing its contents with a value specified by a jump instruction or incrementing it an extra time as in a test skip instruction. All arithmetic and logical operations are performed on operands in an accumulator with results appearing in an accumulator. That is, the machine is not capable of performing these operations directly between memory locations or between a memory location and an accumulator. Each accumulator has an associated carry. After an operation has been completed, a secondary operation may be specified, such as: swap left and right halves of the accumulator, test the contents of the accumulator for a skip, or combine the accumulator with its carry and rotate once left or right.

The I/O hardware supports both a cassette tape transport with two drives and an ASR-33 teletype with papertape punch and reader. The cassette system stores digital information on a single track .15-inch magnetic tape. Data is transferred at 800 words per second between the computer and the transport via the computer's data channel. Each transport spaces (counts records) forward or reverse at 30 inches/sec and rewinds completely in 85 sec. The control writes onto each tape in groups of bits called words, groups of words called records, and groups of records called files. Words are 16-bits long, records are from 1 to 4096 words long, and files are from 2 to 4096 records long. Records are terminated by a 16-bit cyclical checkword and a $1\frac{1}{2}$ " gap;

files are terminated by two 1½" gaps and an End of File (EOF) marker. The tape begins and ends with a 22" leader/trailer.

The ASR-33 teletypewriter is really four devices: keyboard, printer, reader and punch. The teletype separates its input and output functions into two distinct classes. Each class has its own device code, its own Busy, Done and Interrupt Disable flags, and its own interrupt priority mask assignment. To output a character, the code for the character is placed in the output buffer and the Output Busy flag is set. When the teletypewriter is available, it will take the output information and process it. To input a character, the appropriate key is struck placing the corresponding code into the input buffer where it is retrieved by the program.

A single instruction can transfer a word between an accumulator and a device, and at the same time, control the device operation. The interrupt system facilitates processor control over I/O devices by allowing any device to interrupt normal program flow on a priority basis. The processor acknowledges interrupt by storing PC in location 0 and executing the instruction addressed by the contents of location 1.

B. THE INSTRUCTION SET

The instruction set is comprised of three basic types:

1. Memory Reference Instructions to provide movement of the program to various memory locations.

2. Arithmetic and Logical Instructions for operations on data.
3. Input/Output Instructions for peripheral control.
1. Memory Reference Instructions

The formats for both types of memory reference instruction are shown in Figure 2.1.

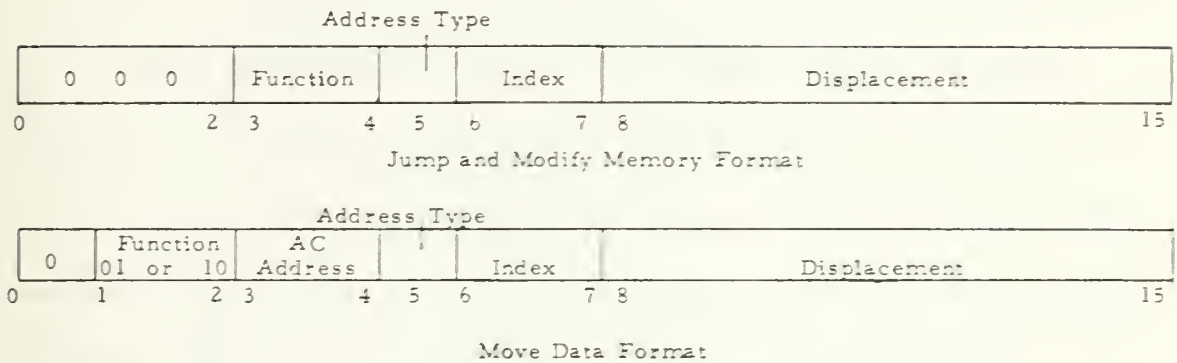


Figure 2.1 Memory Reference Instruction Formats

Fundamental to a memory reference instruction is the manner in which the Nova determines the effective address. As shown in Figure 2.1, both types of instruction are identical in bits 5 - 15. Bits 8 - 15 indicate the displacement (D) and can address $2^8 = 256$ possible locations. Since the displacement is a signed or unsigned number, it has the possible values of $0-400_8$ or -200_8 to $+177_8$. The interpretation of the displacement value is dependent upon the value of the index X (bits 6&7). The index bits are decoded as follows:

X	Address
00	Page zero addressing. D is an unsigned displacement, and indicates an address from 00000-00377.
01	Relative addressing. D is a signed displacement (-200 to +177) that is added to PC.
10	Base register addressing. D is a signed displacement that is added to contents of AC2.
11	Base register addressing. D is a signed displacement that is added to the contents of AC3.
Bit 5 indicates whether the addressing is direct or indirect.	

If bit 5 is a 0, the addressing is direct and the address specified by X and D is an operand, and the instruction can directly address 1024 locations; 256 on page zero and three groups of 256 in the octal range $[PC-200_8, PC+177_8]$, or $[AC-200_8, AC2+177_8]$, or $[AC3-200_8, AC3+177_8]$. If bit 5 is a 1, the addressing is indirect and the address specified by X and D is another address.

In the Jump and Modify Memory format, bits 3&4 indicate the function of the instruction and are decoded as follows:

Function

00	Jump
01	Jump to subroutine, return address in AC3
10	Increment and skip the next instruction if zero
11	Decrement and skip the next instruction if zero

For the Move Data Instruction, bits 3 and 4 indicate the accumulator used by giving the binary equivalent of the accumulator number as follows:

<u>Bits 3&4</u>	<u>Accumulator</u>
00	AC0
01	AC1
10	AC2
11	AC3

In the Move Data Instruction bits 1 and 2 indicate the function and are decoded as follows:

Function	
01	Load accumulator
10	Store accumulator

One additional feature is the auto-incrementing locations 00020-00027 and the auto-decrementing locations 00030-00037. If at any level in the effective address calculation, an address word is fetched from one of these locations, the word is either incremented or decremented by one before the fetch is made.

2. Arithmetic and Logical Instructions

The format for these instructions is shown in Figure 2.2.

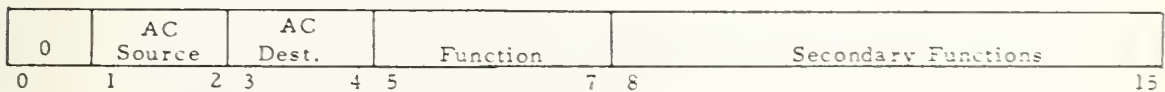


Figure 2.2 Arithmetic and Logic Instruction Formats

Bits 1&2 specify the source AC and bits 3&4 specify the destination AC, that is, where the answer to the specific function will reside. Bits 5-7 are the function or OP code. Bits 8-15 indicate the secondary function to be performed, and are decoded as follows:

Bit 8-9	Shift Operation
00	None
01	Combine the destination AC with its carry and shift left one bit. This operation is cyclic and the carry becomes bit 15
10	Same as above except the operand is shifted right one bit. Bit 15 becomes the carry
11	Swap halves of result. Bits 0-7 are swapped with bits 8-15. Carry remains unchanged.
Bit 10-11	Base Value of Carry Bit
00	Current value of carry
01	Zero
10	One
11	Complement the current value of carry
Bit 12	No Load
0	Normal instruction execution
1	Destination address is not loaded with result
Bit 13-15	Skip Condition
000	Never skip
001	Always skip
010	Skip on zero carry

011	Skip on non-zero carry
100	Skip on zero result
101	Skip on non-zero result
110	Skip if either carry or result is zero
111	Skip if both carry and result are zero

The arithmetic and logical instruction formed by the preceding bits functions to control the arithmetic unit. A diagram of the unit is shown in Figure 2.3.

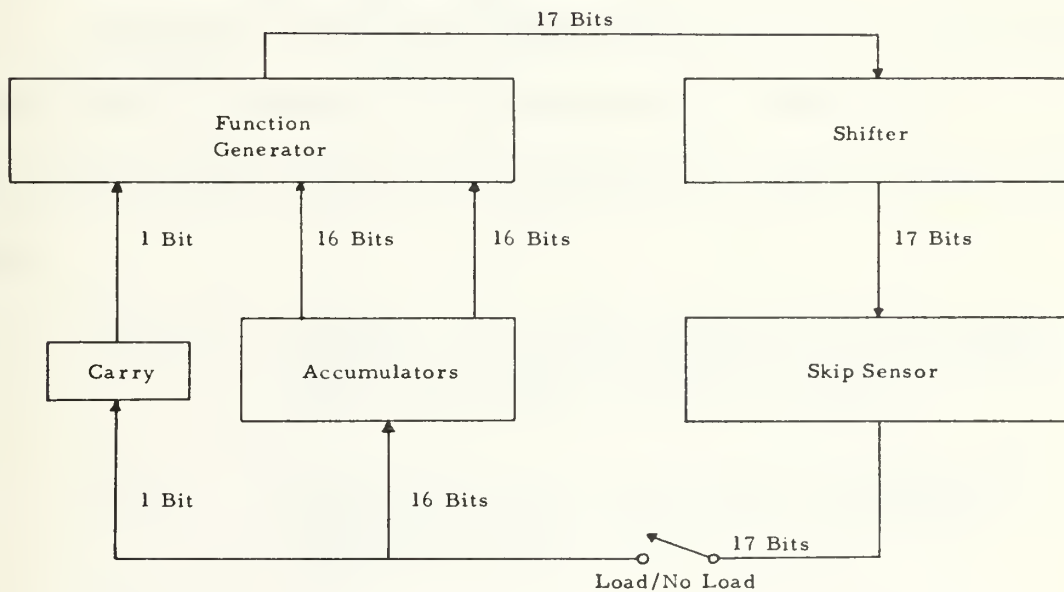


Figure 2.3 Organization of Arithmetic Unit

Each instruction specifies the number of accumulators needed for the specific function. The OP code controls the operation of the function generator. The output of the function generator, consisting of the carry and the 16-bit result is sent to the shifter and skip sensor for possible operation. The determination of load and no-load is made last so that the function can be performed, shifted and/or tested for skip, and still not loaded to the destination AC. This is useful if the operation is only to be used as a test and the previous value in the destination AC or the previous value of the carry is not to be changed.

a. Special Arithmetic Instructions

An additional option available on this model of the Nova 800 is a one-instruction multiply and divide command. These two instructions operate only on unsigned binary numbers. Their operation is as follows:

- Multiply -Multiply AC1 by AC2, add product to AC0, put result in AC0-AC1. AC0 will contain the high order bits and AC1 the low order bits.

- Divide -If overflow, set carry to one. Otherwise divide AC0-AC1 by AC2. Put quotient in AC1 and remainder in AC0.

3. I/O Instructions

The format for this type of instruction is shown in Figure 2.4.

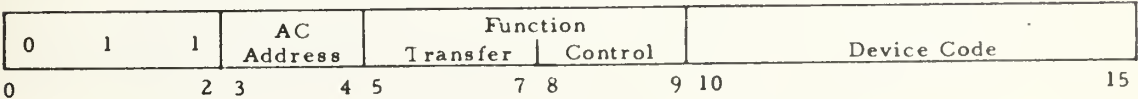


Figure 2.4 I/O Instruction Format

Bits 3&4 select the accumulator that contains the data for output or will receive the input data. Bits 5-9 specify whether the instruction is for input or output, to or from what buffer of the peripheral the data is transferred, and also control the peripheral. Bits 5-7 control the transfer and are coded as follows:

Bits 5-7	Transfer
000	No transfer
001	Data in A Buffer
010	Data out A Buffer
011	Data in B Buffer
100	Data out B Buffer
101	Data in C Buffer
110	Data out C Buffer
111	Skip condition

If bits 5-7 are all 1's then bits 8-9 specify a skip condition as follows:

Bits 8-9	Skip Condition
00	Skip next instruction if Busy is Non-zero
01	Skip next instruction if Busy is Zero
10	Skip next instruction if Done is Non-zero
11	Skip next instruction if Done is Zero

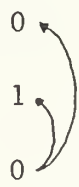
If bits 5-7 are anything other than all 1's then bits 8-9 specify a device control and are decoded as follows:

Bits 8-9	Control
00	None
01	Start device by clearing Done and setting Busy
10	Clear both Done and Busy; i. e., idle the device
11	Pulse a special I/O bus control line if used by the device

Bits 10-15 specify the device code.

The above instructions are used to transfer data between the computer and a peripheral and at the same time control the peripheral by use of the Busy and Done flags. These flags together indicate the state of the device. If both flags are clear, the device is idle and can be placed in operation by setting Busy. After the device has completed an operation, it sets Done and clears Busy to indicate that it is ready to receive new data or has additional data for input. After a complete I/O operation has taken place, the program clears Done and the device is again placed in the idle condition. The normal I/O sequence is shown below:

<u>Condition</u>	<u>Busy</u>	<u>Done</u>
IDLE	0	0
START	1	1
Device Completion	0	0



4. Mnemonics

Normally the programmer is not directly concerned with the bit positions and functions for a given instruction. This is because the

computer is equipped with an assembler which transforms code words called "mnemonics" into the proper bit sequence. But, to fully appreciate the flexibility and capability of the machine, each programmer should familiarize himself with these details, for in them lie the information needed to use the computer most effectively. Instructions for programming by use of mnemonics is given in Section F of this chapter and Reference [D-8]. A complete listing of the entire mnemonic instruction set is in Appendix A.

C. DATA CHANNEL

Handling data transfers between external devices and memory under program control requires an interrupt and the execution of several instructions for each word transferred. To allow greater transfer rates, the processor contains a data channel through which a device, at its own request, can gain access to memory. Besides the straightforward transfer of a word, the channel allows a device to increment by one a word already in memory. This allows an input data stream to be placed in consecutive data locations.

The program cannot affect the data channel directly because there are no instructions for it; instead, the program sets up the device to use the channel. When a device requires access, it requests it from the processor. At the beginning of every memory cycle the processor synchronizes any requests that are being made. The processor is

capable of operating at two different speeds (standard and high speed) and does not require the device to wait until completion of an instruction - the processor can pause to handle transfers at certain points within an instruction. When the processor pauses, it handles all data channel requests, high speed requests first, and then continues with the interrupted instruction. The maximum data rate for standard speed is 500,000 words per second, and for the high speed is 1,250,000 words per second; but at this rate, all other processing activity is suspended.

D. SPECIAL OPTIONS

In addition to the multiply and divide instructions previously discussed, the Nova has another option which is especially useful in data acquisition. This is the Real Time Clock (RTC). The clock is controlled by use of one of the I/O instructions. The mnemonic is "DOA X, RTC", where DOA is the Data Out, Buffer A instruction acting on the RTC. X indicates the accumulator where the frequency data is held. Bits 14-15 of X control the frequency of the clock and are decoded as follows:

Bit 14-15	Frequency
00	Ac line frequency
01	10 HZ
10	100 HZ
11	1000 HZ

E. THE OPERATING SYSTEM

To program the Nova, there are six basic sections of software which must be used to build an instruction sequence. The Self-loading Bootstrap and Image Core Loader/Writer provide the initial loading capability. The Command Line Interpreter implements mnemonic loading of utility programs and performs file maintenance chores. The Text Editor is used to build and change a mnemonic program. The Assembler converts the mnemonic source program into a binary language suitable for computer operation. Finally, the Relocatable Loader performs the actual program load.

1. Self-loading Bootstrap and Image Core Loader/Writer

Before a program can be brought into memory, the loading program must reside in memory. In other words, the computer must be programmed so that it has a loading capability. This is the function of the self-loading bootstrap. The bootstrap is resident in the hardware of the machine on ready-only LSI chips and is initiated when the program load switch is momentarily placed in the up position. The bootstrap program is placed in location 0-37 by the processor and normal operation is begun at location 0. First the data switches on the front panel are read to determine the I/O device code used and whether or not a data channel is required. Once this information is processed, normal loading will occur starting in location 0 while the loader scans location 377 to determine when a data word has been loaded. When a word is

loaded into location 377, it is executed as an instruction. Normally this instruction is a jump to the data just read or simply a halt.

This allows the loading of a program of limited size (256 words) and so the program loaded must itself be a loading program. This is normally the image core loader/writer. The loader now takes program control and loads an image of itself into the high 377₈ locations in core. If these locations are left undisturbed, either the loader or writer may be recalled at any time. At the end of the loading process, the loader outputs a prompt "#" on the teletype to indicate that it is ready to accept a command. If during any computer operation the loader is needed, it may be recalled by setting the data switches to the highest address in core, pressing RESET and then START. If the writer is needed, it may be called by setting the data switches to the second highest address in core, pressing RESET and then START.

If either the loader or the writer encounters an error during its operation, "*ERR" will be output to the teletype and the error status word will be placed in ACØ. Additional information is available in Ref.

[D-2]. The error status codes are given below:

<u>Bit</u>	<u>Meaning</u>
1	Data Late
3	Illegal Command
5	Lateral Parity error in a word
6	Addressed tape is beyond the End of Tape (EOT) marker

- 8 Addressed tape is at a load point
- 10 Bad tape
- 13 Unit is write locked
- 14 Odd number of bytes detected in a read or write attempt

2. Command Line Interpreter (CLI)

The CLI implements mnemonic loading of other utility programs from a master cassette tape and performs certain file maintenance chores. When the CLI is ready to accept a command, it issues a prompt "R" followed by a carriage return.

In order to use the CLI, the core image loader/writer should be resident in core and the master tape should be loaded on cassette drive 0.

The format of this master tape is as follows:

- File 0: Core image loader/writer (bootstrapable)
- 1: Command line interpreter
- 2: Extended Assembler (ASM)
- 4: Extended Relocatable loader (RLDR)
- 5: Library file edition (LFE)
- 6: SYSGEN (SYSG)

Normally when a utility program is loaded, the CLI is overwritten and therefore must be reloaded to be used again. Any of files 2-6 may be loaded by typing their code name followed by a carriage return. The following file maintenance operations may also be performed:

1. Load an Absolute Binary Paper Tape

Format: BLDR \$TTR

The CLI will prompt the user to load the paper tape reader and strike any key on the teletype. The tape is then read into memory.

2. Load a cassette file

Format: CTX:YY

where: X is the cassette drive number (0-7)
YY is the file number on that drive (0-99)

For this command the cassette must have already been loaded on the cassette drive and the format of the tape must be core image. The definition of the core image format will be covered under the section on the Extended Relocatable Loader. This is the method for loading a program which has already been assembled and saved for future use.

3. Make a save file

Format: MKSAVE absolute-binary-filename
output filename

The input absolute binary file is converted to a core image (save) file and stored. For example:

MKSAVE CT0:1 CT1:2

The absolute file on cassette drive 0, file 1 is converted to core image and stored on cassette drive 1, file 2.

4. Transfer a file

Format: XFER source filename
destination filename

This command copies a complete binary file and stores it at the destination filename. No conversion of format is done and the source file is not destroyed.

It should be noted at this time that if the user only wishes to use the CLI to load another utility program, time can be saved by simply calling the specified file while in the Core Image Loader/Writer mode. For example, to load the CLI the user types 1 and carriage return. This is because the CLI is resident in file 1 of the master tape. But if the user intends to go directly to the Text editor, he may do so by typing 2 and carriage return and bypass the CLI altogether. The same is true for any of the other utility files.

Ref. [D-3] contains less frequently used commands and some additional information on the CLI. Possible error messages when using the CLI are listed below.

ERROR

NOT ENOUGH ARGUMENTS

ILLEGAL FILE NAME

ILLEGAL COMMAND FOR DEVICE

DEVICE IS READ PROTECTED

CHECKSUM ERROR

FILE NON-EXISTENT

PHASE ERROR

3. Text Editor

Essentially the text editor has two functions: 1) building a user mnemonic program, 2) changing a program which already exists. Ref. [D-4] is fairly detailed in the use of the editor and therefore the material presented here will amplify some points of possible confusion presented in that text.

When the text editor is loaded and ready to accept a command it prompts the user by outputting an asterisk (*) onto the teletype. At this point, the user issues a command or string of commands using the ESC (escape) key on the teletype in much the same manner as the carriage return is used for other programs. When the ESC key is hit, a "\$" is echoed to the teletype. Example:

1. Single Command

```
GRCT1:1$
```

This command opens the file on cassette drive 1, file 1 for reading into the input buffer. If this command is to be executed, the character string GRCT1:1\$\$ is typed; the final \$ causes execution to begin.

2. Command Strings

```
GRCT1:1$Y$B$T$$
```

This string first opens the same file as in example 1, but then continues by yanking the first page into the buffer, placing the character pointer at the beginning of the buffer

and finally typing the entire buffer on the teletype.

Notice that a single ESC (\$) separates the commands but two consecutive \$'s will prompt the editor to begin execution.

To start an input file, the letter "I" is typed directly following the prompt (*) by the editor. After this a complete page of input (256 lines or less) may be typed and will be stored in the editor buffer. The end of this input string must be followed by a double ESC (\$\$). The "I" will not be placed into the buffer but all characters between "I" and "\$\$" will.

The reference is in error on its description of the use of a tab. It states that a tab may be executed by typing "P" while depressing the CTRL key. In the present version of the text editor, tab is executed by typing "I" while depressing the CTRL key. The CTRL P key combination deletes the tab capability from the text editor and it may only be restored by reloading, so this key combination should be avoided.

When a file is to be punched onto an output file, it must be accompanied by a form feed. This is used by the computer to recognize where one file stops and another begins. The reference states that the punch command (P\$) includes a form feed but this is not the case. So once a punch command is given, it must be followed by a form feed command (GC\$). For example:

GWCT0:2\$B\$P\$GC\$\$

This command string opens cassette drive 0, file 2 for the writing, places the character pointer at the beginning of the buffer, writes the buffer onto the file and places a form feed at the end.

A word of caution on cassette file organization is needed at this point. File manipulation is not taken care of by the software as it is in larger computers. It is the responsibility of the user. So if information is stored on files 0-4 and file 2 is to be changed, the contents of files 3 and 4 must be saved on a scratch tape until the change is made and then placed back on the tape. If this is not done, the ordered structure of the tape will be destroyed and the information on files 3 and 4 will be lost. Another problem occurs when the user tries to transfer information from one file to another on the same drive. Again, due to hardware limitation, this cannot be done directly. The file must be transferred to a scratch tape and then to the destination file.

Finally, the reference for the text editor should be studied carefully to gain a good working knowledge of the operation since it is the utility routine that is used most often. Each type of command should be tried to completely understand what it actually does.

POSSIBLE ERROR MESSAGES

NOT ENOUGH ARGUMENTS

OUTPUT FILE WRITE PROTECTED, FILE: filename

NO OUTPUT FILE SPECIFIED

ILLEGAL SYMBOL NAME: symbol name, (Invalid character in command line)

FILE DOES NOT EXIST, FILE: filename

UNEXPECTED SYSTEM ERROR (Computer halts with the system
error codes AC2)

4. Assembler

As with the text editor, much of the required information on the assembler is provided in References [D-5] and [D-6]. The information presented here represents the basic considerations needed for proper program control and does not attempt to describe the details of the routine.

When a program is written, the user has three options as to where the program should reside in core. These are 1) absolute location where the actual core location of the program is specified, 2) relocatable location where the actual location is controlled by the loader which places the program in any convenient location not already occupied, 3) a combination of 1) and 2) where some portion of the program is in a definite location but other portions are left to the loader.

If a program is to reside in a specific location, a .LOC XXXXX statement is placed prior to the start of the program. For example:

```
.LOC 400  
PGM:    LDA    2,1  
        .  
        .  
        .  
        .END
```


This sequence causes the program named PGM to be placed in core starting at address 00400_8 .

If the core position is to be left to the loader, the user simply places the statement .NREL prior to the start, that is:

```
.NREL  
  
PGM:      LDA    2,1  
  
.  
  
.  
  
.  
  
.END
```

If a location is to be specified on page zero (location $0-400_8$) but the position of that page may be placed anywhere in memory, the statement .ZREL should be placed prior to a .LOC statement. For example:

```
.ZREL  
  
.LOC    400  
  
PGM:      LDA    2,1  
  
.  
  
.  
  
.  
  
.END
```

In a number of cases there is no real need to specify the location of the entire program but the address of a subroutine is to be placed

at a specific location. In this case the control statement might appear as shown below:

```
                .LOC  40  
  
                APUT  
  
                .NREL  
  
PGM:           LDA   2,1  
  
                .  
  
                .  
  
                .  
  
                .END
```

In this case the address of a subroutine called APUT is placed in location 00040₈ but the main program, PGM and the actual subroutine APUT could be placed anywhere.

The process of linking together programs and/or programs and subroutines is done by the .EXTN and .ENT statements. If a program is a subroutine of the main program, the name of the subroutine must be listed as an external in the main and as an entry point in itself. For example,

```
                .ENT   PGM  
  
                .EXTN  APUT  
  
                .LOC  40  
  
                APUT  
  
                .NREL  
  
                .
```



```
PGM:   LDA      2, 1
```

```
.
```

```
.
```

```
.
```

```
.END   PGM
```

```
.ENT   APUT
```

```
APUT:  STA      3, SAV
```

```
.
```

```
.
```

```
.
```

```
.END   APUT
```

In this example the main (PGM) uses the subroutine (APUT) and therefore APUT is listed as an external in the main and has an entry statement prior to its initial statement. PGM can also be called by another program and therefore has an entry statement prior to its first statement. One other point is made in this example: the program name is placed after the .END statement. If this is not done the loader will load the program and then halt. It is left to the user to know the address of the first statement in the program so that it may be started. If the name is placed after the end statement, at the end of the load, control will be passed to the program automatically and execution will begin.

When the program is ready to be assembled, the assembler is loaded in the normal manner and prompts the user by typing "ASM" on

the teletype. Although there are several types of assemblies that can take place, only the most common is presented here. The user responds with the following:

```
ASM 1 input filename output filename / assembled format /  
(machine prompt) optional teletype response
```

For example,

```
ASM 1    CT 1:0    CT0:0/B $TTO/L  (carriage return)
```

The response to this command is to take the source program on cassette drive 1, file 0; assemble it with the output to cassette drive 0, file 0 in binary and list the assembly on the teletype. After the user is familiar with the assembly process, there is no need for a teletype listing, so the command is normally:

```
ASM 1    CT 1:0    CT0:0/B  (carriage return)
```

During the assembly, diagnostic and other errors are typed on the teletype with the appropriate error flag. A list of these flags is given in Appendix E.

5. Relocatable Loader

When all or part of a user program is in relocatable format, the required core assignment is performed by the relocatable loader. The loader is called in the normal manner using the CLI and/or the image core loader/writer and prompts the user by typing "RLDR" on the teletype when it is ready for a command. When the loader operates on an assembled program, it converts the binary format to a core image

format. This format is the one used by the image core loader/writer and allows the program to be swapped to any memory location that is necessary. In larger systems, the program may be swapped in core many times especially if the NOVA were used for multiprocessing. It is for this reason that the core image format was established.

Basically there are two ways to use the loader. First, if the user wishes to know where the program will reside in core, he types,

```
RLDR  filename(s) to be loaded   Save file established/S   $TTO/L
(machine prompt) (carriage return)
```

In using the loader it is always necessary to establish a save file. This file is extremely useful for a program which will be used more than once. Once the save file is established, that program complete with subroutines may be initiated at a future time by merely loading the CLI and performing the loading function prescribed in Section 2. An example of the loader operation is:

```
RLDR  CT0:0  CT0:1  CT0:2  CT1:0/S  $TTO/L (carriage return)
```

This command loads cassette drive 0, files 0, 1, 2; establishes a save file on cassette drive 1, file 0; lists the location of all programs on the teletype. A typical teletype output is:

```
.MAIN
.MAIN
.MAIN
  NMAX 000555
  ZMAX 000050
  CSZE
  EST
  SST
```


PGT	000440
.BIND	000465
APUT	000536

OK

The information contained in this printout is as follows:

1. Three files were loaded as indicated by the three ". MAIN"
2. NMAX indicates the next normal relocatable address available after the load; in this case it is 000555₈
3. ZMAX indicates the next zero relocatable address available after the load; in this case it is 000050₈
4. CSZE, EST, SST have no meaning unless a disk type memory is available.
5. PGT is the name of the program on CT0:0 and it has been placed starting in location 000440₈. Likewise .BIND was on CT0:1 and starts in location 000465₈. Finally APUT was on CT0:2 and starts in location 000536₈.
6. OK indicates that the loading process is completed.

If the user has no need for the core position, he may eliminate that portion of the printout by simply typing the same command without the "\$TTO/L" at the end.

Additional information on the relocatable loader is contained in References [D-3] and [D-7]. Possible error messages are listed below:

ERRORS:

NO INPUT FILE SPECIFIED

NO SAVE FILE SPECIFIED

SAVE FILE IS READ/WRITE PROTECTED

F. GENERAL PROGRAMMING

Reference [D-8] is the primary source for information on programming the NOVA computer. The following are some programs which the author used for machine checkout and familiarization. These programs highlight the major portions of the system which are directly applicable to digital control.

The first step was to become familiar with the subroutines supplied by Data General which have possible application to control problems. These are listed in Appendix F.

Next, a communications link between the computer and the teletype was established. This link must be bi-directional; therefore two subroutines were written. The first routine is called AGET and is used to read a character or number from the teletype into ACØ and echo that same character or number back to the teletype. Bit 8 of the word is masked out since it is not used by the computer and can cause some problems in other routines which use the character. The routine is listed below.


```

                                ;INPUT FROM TTY.
                                ;N INTO AC0
. ENT AGET
. NREL
AGET:  NIOS      TTI      ;READ CHARACTER
      SKPDN     TTI      ;WAIT TILL RECEPTION DONE
      JMP       .-1      ;LOOP
      DIA       0,TTI    ;BRING IN CHARACTER
      SKPBZ     TTO      ;
      JMP       .-1      ;ECHO
      DOAS      0,TTO    ;
      LDA       2,EGT    ;LOAD MASK
      AND       2,0      ;MASK OUT BIT 8
      JMP       0,3      ;RETURN
EGT:   -201
      . END

```

The second routine, called APUT, is used to output a character or a number to the teletype followed by a carriage return and line feed. The carriage return is initiated by the receipt of a null for output. The output is through ACØ. The routine is listed below:

```

                                ;OUTPUT TO TTY. WITH LF AND CR.
                                ; N IN AC0
. ENT APUT
. NREL
APUT:  SKPBZ     TTO      ;WAIT FOR CHARACTER
      JMP       .-1
      DOAS      0,TTO    ;OUTPUT CHARACTER
      MOV       0,0,SZR  ;SKIP IF NULL
      JMP       0,3      ;NULL RETURN
      STA       3,SPUT   ;SAVE RETURN
      LDA       0,CR     ;OUTPUT CR
      JSR       APUT     ;RECURSE
      LDA       0,LF     ;OUTPUT LF
      JSR       APUT     ;RECURSE
      SUB       0,0      ;RESTORE NULL
      JMP       @SPUT    ;NOT NULL RETURN

SPUT:  0
CR:    15          ;ASCII CODE FOR CR
LF:    12          ;ASCII CODE FOR LF
      . END

```


The first program written was to simply add two numbers, convert their sum to decimal and output the result to the teletype. The routine is listed below:

```

                ; ADD TEST
                .ZREL
                .ENT PGM
                .EXTN .BIND, APUT
                .LOC 41
                APUT
BINDE:          .BIND
                .NREL
PGM:            LDA      1, X      ;LOAD AC1 WITH X VALUE
                LDA      2, Y      ;LOAD AC2 WITH Y VALUE
                ADD      2, 1      ;ADD AC2 TO AC1, RESULTS IN AC1
                JSR      @BINDE    ;JUMP TO CONVERSION ROUTINE
X:              5
Y:              6
                .END PGM
```

This program highlights some of the linking considerations which must be made in order to successfully combine programs. First the program name (PGM) is declared as an entry point so that a return can be made by the subroutines. Next the binary to decimal routine (.BIND) and the teletype output routine (APUT) are declared as normal external names. The address of APUT is placed in location 00041₈ because .BIND makes an indirect jump through this address. Finally, the address of .BIND is assigned to the name BINDE. In this way, the Jump Subroutine Command (JSR) jumps indirectly through BINDE to the location of .BIND. This is not the only method for linking programs but by adhering to the following rules, very few problems were encountered.

1. Declare the program name as an entry point.
 - a) Some programs may use more than one entry point so include all the points whether you intend to use them or not.
2. Declare all other programs and subroutines as normal externals whether the link is directly from the main program or only between secondary programs.
3. Assign the name of secondary programs and subroutines another name and use the indirect JSR command.

Example: JSR @BINDE
 not
 JSR .BIND

The next program to be written was to test the operation of the real time clock. The test procedure is to establish a number (400_8) and then to subtract one from it and write the result on the teletype in decimal. The consecutive subtractions were to take place every 10 seconds. The program is listed below:


```

                                ;CLOCK TEST
      .ZREL
      .ENT PGT
      .EXTN .BIND, APUT
      .LOC 41
      APUT
BINDE: .BIND
      .NREL
PGT:   SUBO      0,0           :CLEAR AC0
      INCS       0,1           ;PLACE 4008 in AC1
      LDA        2,X           ;PLACE X IN AC2
      ADD        2,1           ;ADD X TO 400
      STA        1,HUNT        ;SAVE RESULT
      JSR        @BINDE        ;JUMP TO CONVERSION AND OUTPUT
      LDA        0,DELAY        ;LOAD DELAY COUNT
      STA        0,COUNT        ;SAVE COUNT
      LDA        0,HZ1         ;LOAD HZ1 IN AC0
      DOAS       0,RTC          ;START REAL TIME CLOCK
      SKPDN      RTC           ;WAIT FOR FIRST CLOCK PULSE
      JMP        .-1           ;
      DSZ        COUNT         ;DECREMENT COUNT BY 1,
                                ;SKIP IF RESULT ZERO
      JMP        .-4           ;RESTART CLOCK
      LDA        1,HUNT        ;TIME EXPIRED, LOAD RESULT
                                ;INTO AC1
      JMP        .-15          ;RESTART

DELAY: 144
COUNT: 0
HZ1:    1
X:      -1
HUNT:   0
      .END      PGT

```

The major points in this program are those involving the operation of the clock. The DOAS 0, RTC command loads the contents of AC0 into buffer A and then starts the clock. In this case, the clock frequency called for is 10 Hertz. The clock frequency selected in the program gives an output pulse every 0.1 sec. These pulses decrement a counter which is initialized to 100 (144 octal). When the counter is

zero, 10 seconds have elapsed and the program is restarted. During the execution of this program it was found that the time duration between outputs was nearly 11 seconds. This is attributed to the amount of computations between the initiation of the program and the start of the delay sequence. That is, the first six instructions in the main and the time required for .BIND and APUT. APUT requires some time due to the slow reaction of the teletype and .BIND is fairly lengthy. Compensation for this delay can be made by adjusting both the clock frequency and the delay count.

One final item should be mentioned at this time. Throughout the previous programs, various names have been used for constants and indirect addresses. The use of a name is restricted if the assembler recognizes it as a special command used in a number of different operating systems. These names are listed on system parameter tapes and have been compiled into one listing and placed in a binder. The user should familiarize himself with this listing to avoid assembly errors caused by using restricted names.

III. DATA ACQUISITION AND DISTRIBUTION

A. INTRODUCTION

Chapter 2 provided some information on the computer, but this is only half of the system. In this chapter, consideration is given to the devices used to transform analog information into a digital word and vice versa. In the past, this task was accomplished by the use of external equipment which was comparable in size to the computer itself. But with recent strides in LSI, manufacturers are beginning to market multichannel A/D and D/A equipment small enough to be placed inside the computer mainframe and mounted on a general purpose interface board.

The initial considerations were that of resolution, speed and size. With these specifications set, the devices were selected and the interface problem began. Interface considerations were very detailed and required a thorough understanding of the interface equipment and the devices to be adapted. Of the devices, the A/D converter is most complex, and this chapter presupposes a certain knowledge of this process. A more fundamental approach to A/D and D/A conversion is contained in Appendix B.

B. RESOLUTION AND SPEED COSTS

When developing a system, one of the major parameters is the cost of resolution. As the graph of Figure 3.1 shows, costs rise

fairly slowly up to 12 bits. Above 12 bits, however, they rise rapidly. The price of a 14-bit system is 70% higher than the 12-bit system. A 12-bit system provides a reasonable compromise between resolution and cost, and most physical parameters can be covered adequately by a 12-bit converter. In fact most transducers are hard pressed to achieve linearity better than 0.05%, and a 12-bit system corresponds to 0.012%. It is estimated that the 12-bit converter can cover 80% of the applications that exist today and that most of the others can be satisfied by adding a dynamic-ranging capability.

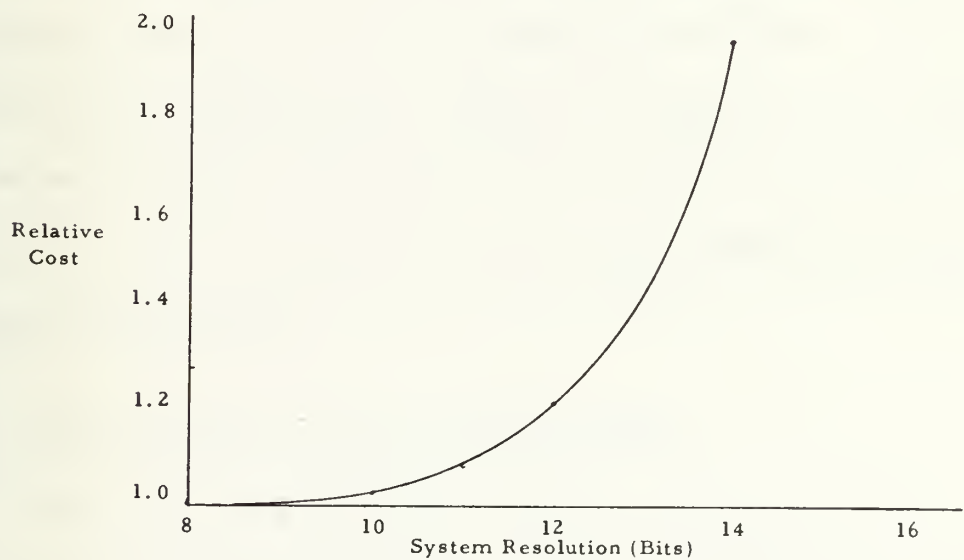


Figure 3.1 Relative Cost vs Resolution

The same sort of reasoning applies to speed. Here the breakpoint occurs at about 100KHZ. Above this frequency an increase in throughput rate becomes quite expensive. Of course this might be justified, but only if the high data rate can be used. For example, if the computer

takes 20 usec to make a conversion, then the system takes 30 usec for each data point. If the speed of the front end were doubled, only an overall speed increase of 16.7% is gained. On the other hand, if the computer can operate in the direct memory access (DMA) mode, it might make sense to increase the speed of conversion since DMA might lower the memory access time to about 2 usec.

For automatic control problems, a 12-bit converter is completely reasonable. The linearity is well within any normal transducer range and if a limit of ± 10 volts is placed on the inputs, the least significant bit (LSB) corresponds to $\pm 0.0048V$ which is well below the minimum response voltage of most analog amplification devices. The speed of the device is based upon the smallest time constant that is expected to be encountered in the control system. Since most of these systems are slow from an electronic standpoint, only a 50KHZ throughput was considered necessary.

C. THE INTERNAL ANALOG PERIPHERAL

Advances in semiconductor technology have brought about the option of building the entire acquisition system in a single package small enough to fit onto the interface board found in most minicomputers. This approach saves both money and space. Costs are saved by the elimination of a separate chassis, power supply, and interconnecting hardware. Also, cabling is eliminated. It is possible to save money

by eliminating the need for a separate power supply since a solid state system shouldn't draw more than about 500mA at 5-volts.

A final reason for going the built-in route is reliability. Connectors are normally the weakest link in any electronic assembly, so by eliminating them we should be able to improve the reliability of the system.

Of course, there are some problems associated with the built-in peripheral. The main one is having an analog device working in a digital environment. There are three major sources of noise in a computer: high frequency clocks, the common for the 5-volt power supply, and magnetic core. Of these, clock and memory noise can be averted by shielding the system (i. e., placing it in a metal can). The noise on the 5-volt power supply line is somewhat more difficult to reduce. For example: a 2.5-volt change in 10 nanosec will couple a 25-millivolt error spike onto an analog input line if the two lines only have a 1-picofarad capacitor between them, and if the source impedance of the analog signal is 100 ohms. Higher source impedance and higher capacitance make the problem worse.

The major way to reduce power supply noise is to make analog runs as short as possible. Analog lines should be routed away from digital lines, shielded, and run in twisted pairs to reduce magnetic pickup.

D. GENERAL INTERFACE EQUIPMENT

The general interface boards supplied with the computer include all possible options for the interfacing peripheral devices. The options used in this application are shown in Figures 3.2 through 3.5.

Figure 3.2 is the data bus control circuit. It is a bi-directional circuit which handles data flow between the processor and the peripheral. Input data is placed on the nand gate input pins and transfer to the processor is controlled by the OR gate U28. Output data is placed on the bus by the processor and is available at the output of the inverters.

Figure 3.3 is a simplified schematic of the control signal inverters. Input transfers are initiated by the DAT1A, DAT1B and DAT1C control lines. Output transfers are initiated by the DAT0A, DAT0B, and DAT0C control lines. Additional information on these and the other control signals is contained in Appendix C. Each inverter is controlled by the device select line. This inhibits the control signals unless the device on that board has been previously selected.

Figure 3.4 is the device select circuit. It is a six-input nand gate which may be strapped so that it responds only to the device code that corresponds to the device on that interface board.

Figure 3.5 is the Done and Busy flipflop control circuit. The sequence of operation of these flipflops was discussed in Chapter II. Either the IO Reset pulse or the $\overline{\text{CLEAR}}$ pulse will place the device in the IDLE mode. The $\overline{\text{START}}$ pulse will place the device in the START

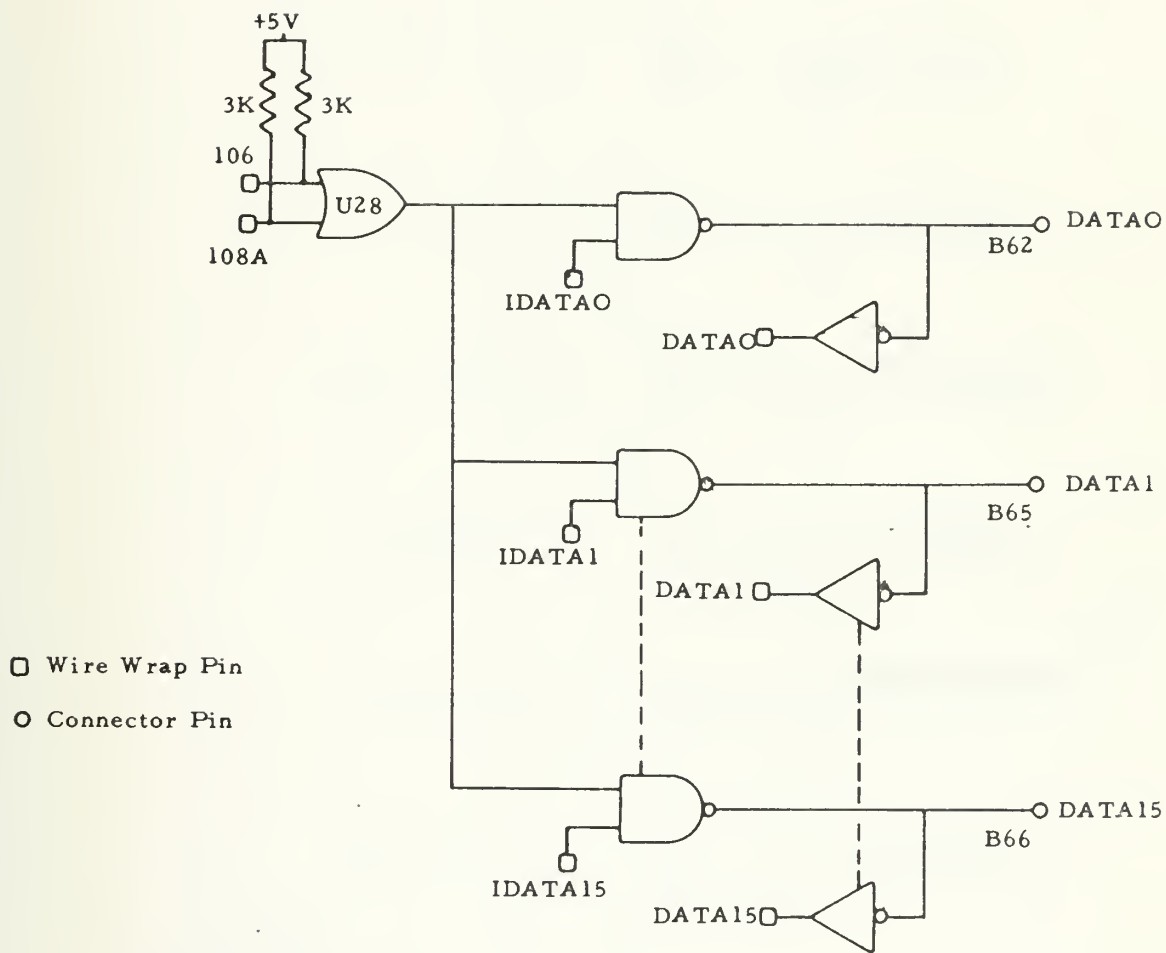


Figure 3.2 Data Bus Control Circuit

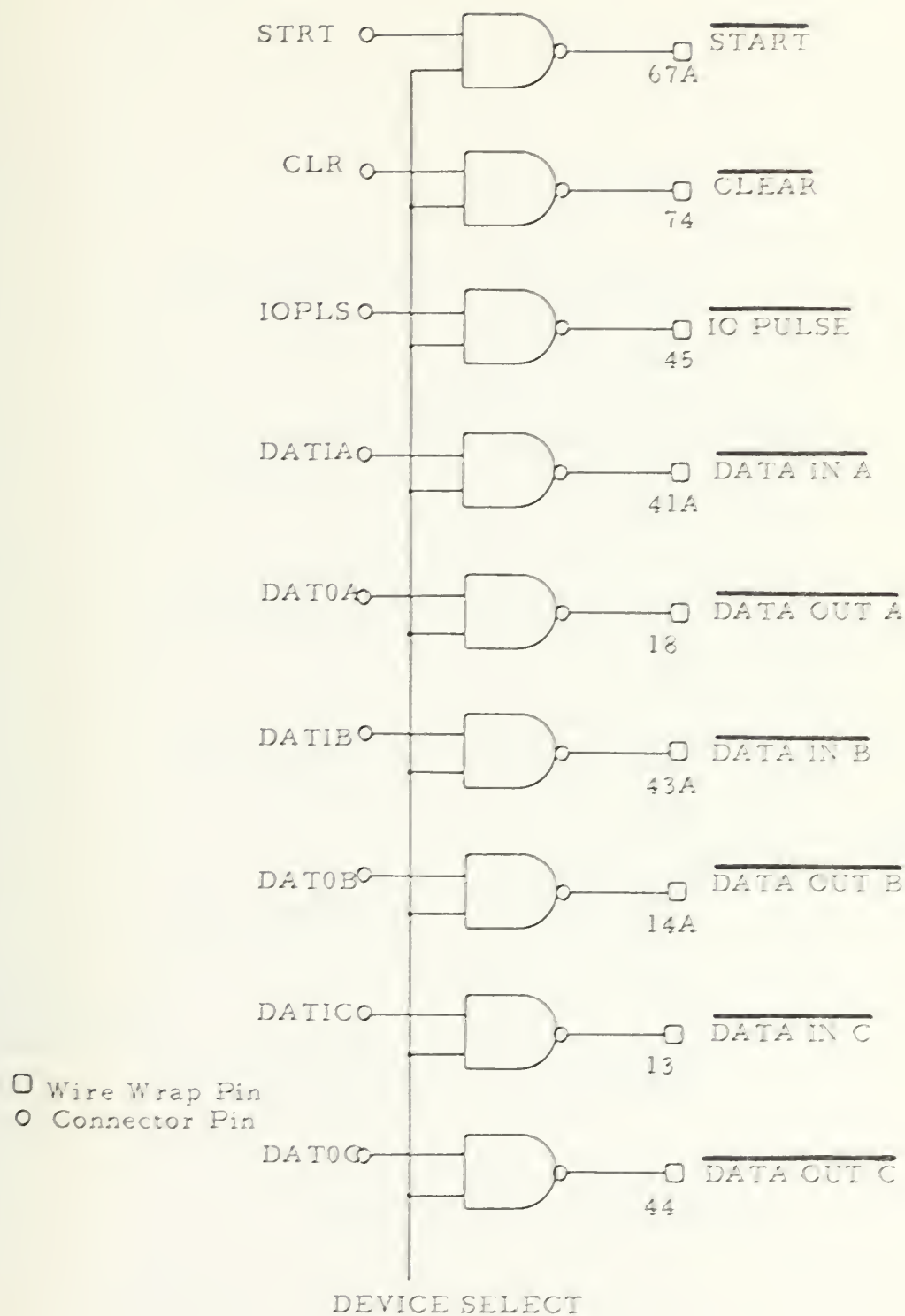


Figure 3.3 Control Signal Inverters

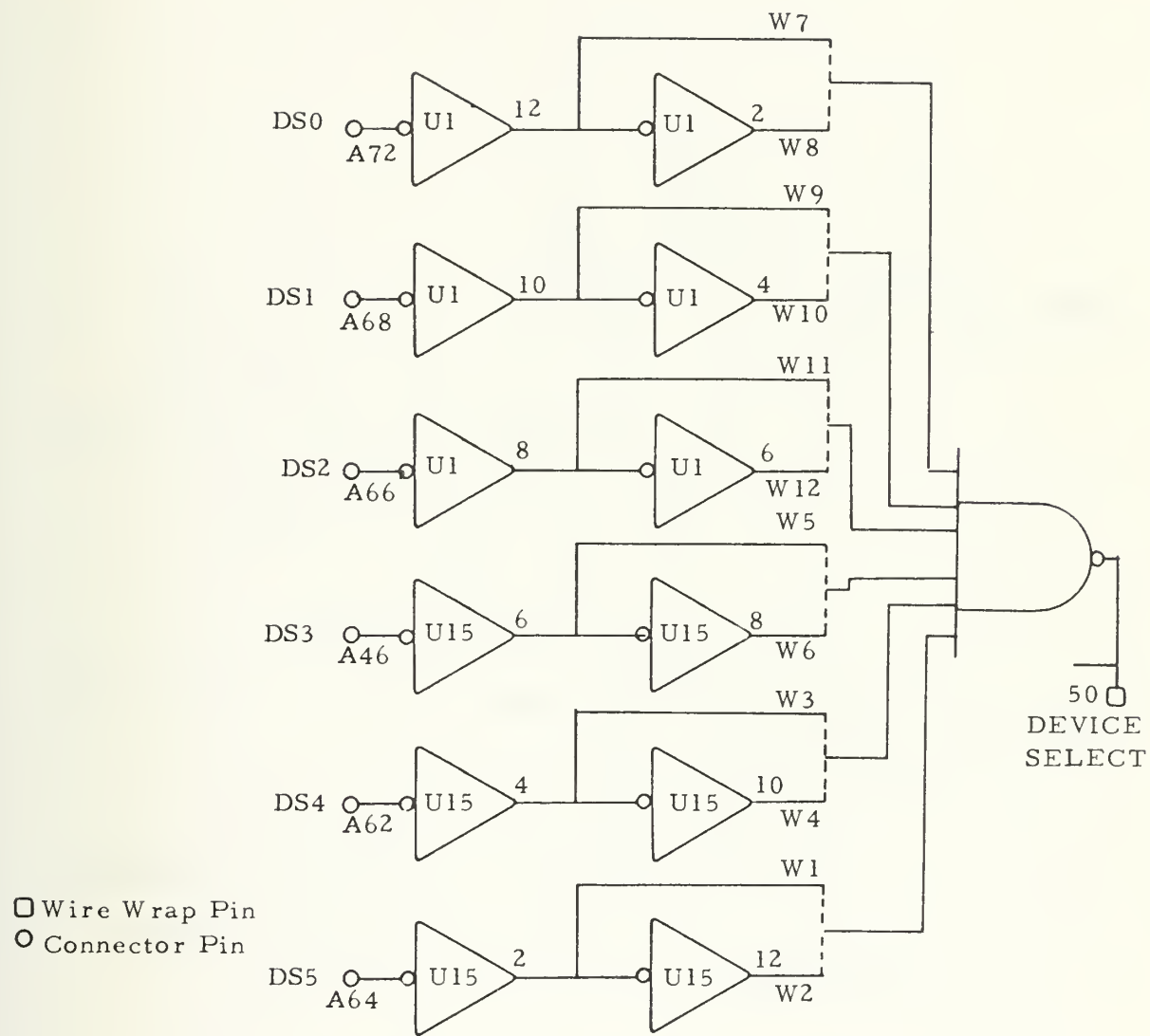


Figure 3.4 Device Select Circuit

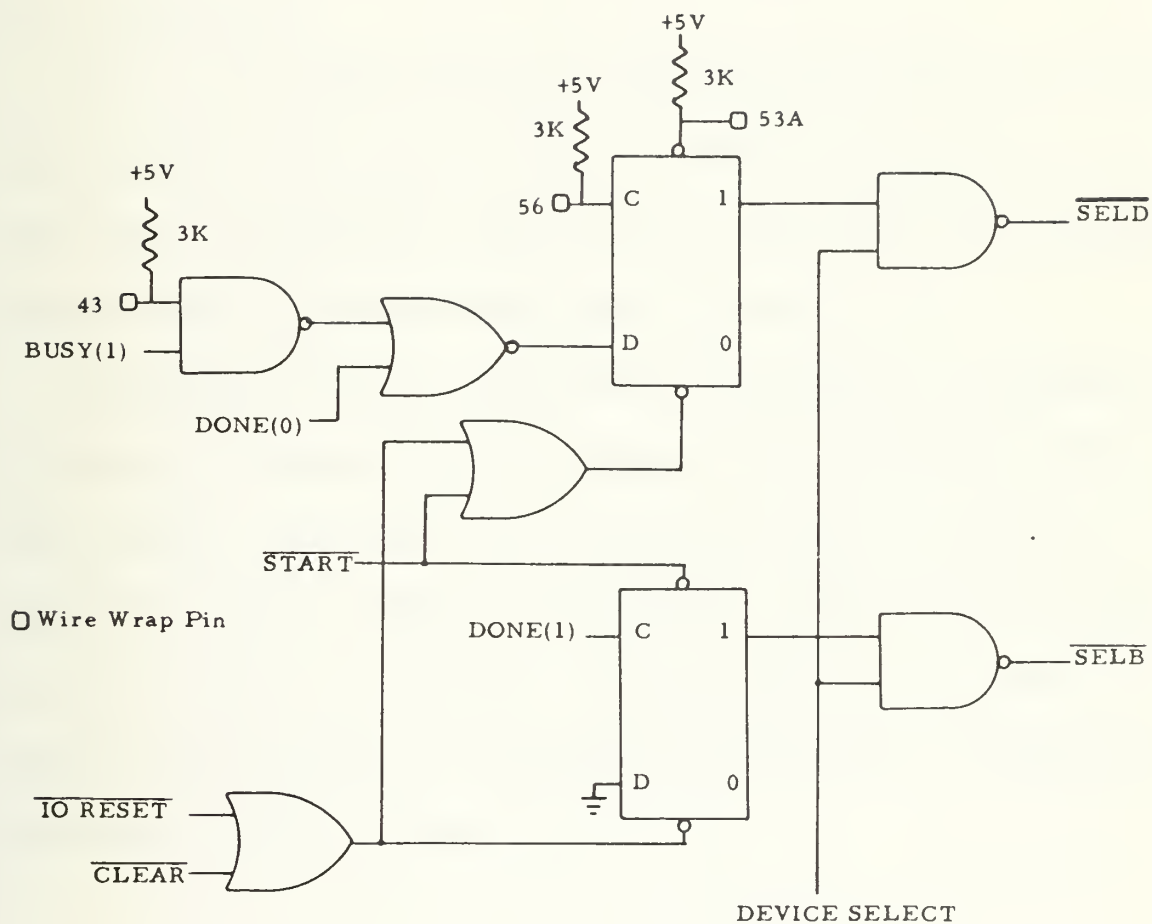


Figure 3.5 DONE and BUSY Flipflop Control

mode. $\overline{\text{SELD}}$ and $\overline{\text{SELB}}$ are the signals sampled by the processor to obtain the device status.

Detailed schematics of the above interface circuits is contained in Reference [D-1].

E. THE DT 1620

Upon reviewing the market for an A/D converter which would meet the previously stated requirements, one device in particular appeared to be more than adequate. This was the DT 1620 which has a throughput rate of 50KHZ. It is produced by Data Translation, Inc.

This module is used for high speed data acquisition. Essentially, it consists of a sixteen-channel multiplexer front end feeding a differential amplifier. The output of the differential amplifier is input to an integrator used as a sample and hold device. Finally, the held signal is fed to a 12-bit A/D converter. A logic programmer is used to provide the timing signals for control of the entire module. A simplified schematic is shown in Figure 3.6.

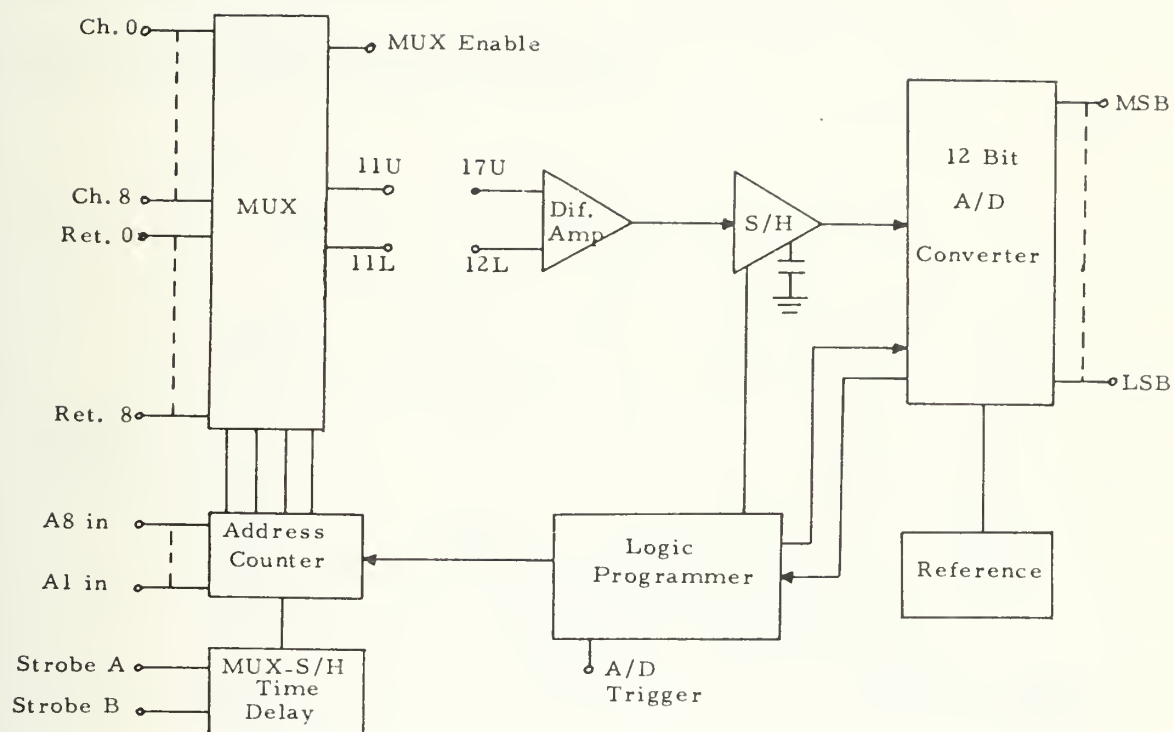


Figure 3.6 DT 1620 Simplified Schematic

1. Multiplexer and Differential Amplifier

The multiplexer is available either in single ended or differential configuration. The choice of these two modes is dependent upon the common mode current. Common mode current normally becomes a problem when measuring voltages which are referenced to different commons. Referring to Figure 3.7 the common mode voltage, V_{cm} , that the differential amplifier sees is represented as a voltage source between the analog return and ground.

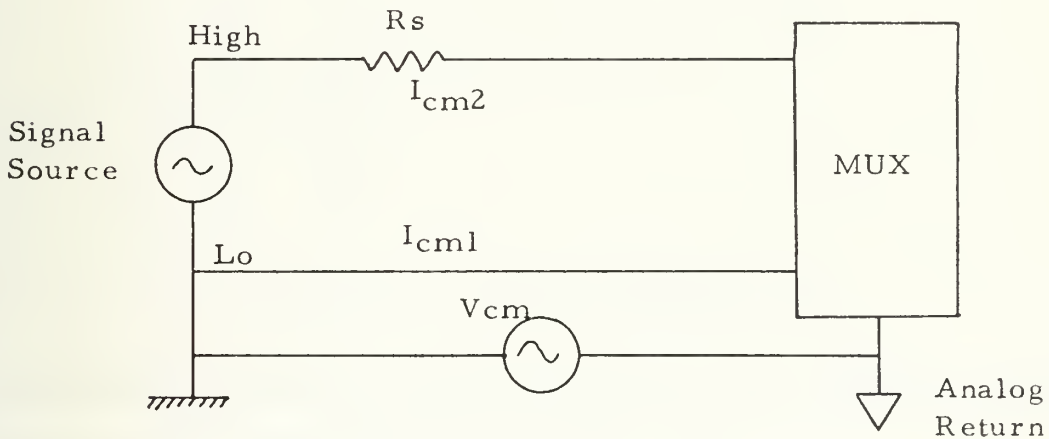


Figure 3.7 Common Mode Voltage Model

The problem created here is that the common mode current passing through the signal source resistance R_s causes a voltage drop that does not exist on the signal L0. If R_s is small (10-20 ohms), the voltage error is negligible, but if the signal sources have different common mode voltages the single ended MUX creates a problem since all signal commons must be tied together as shown in Figure 3.8.

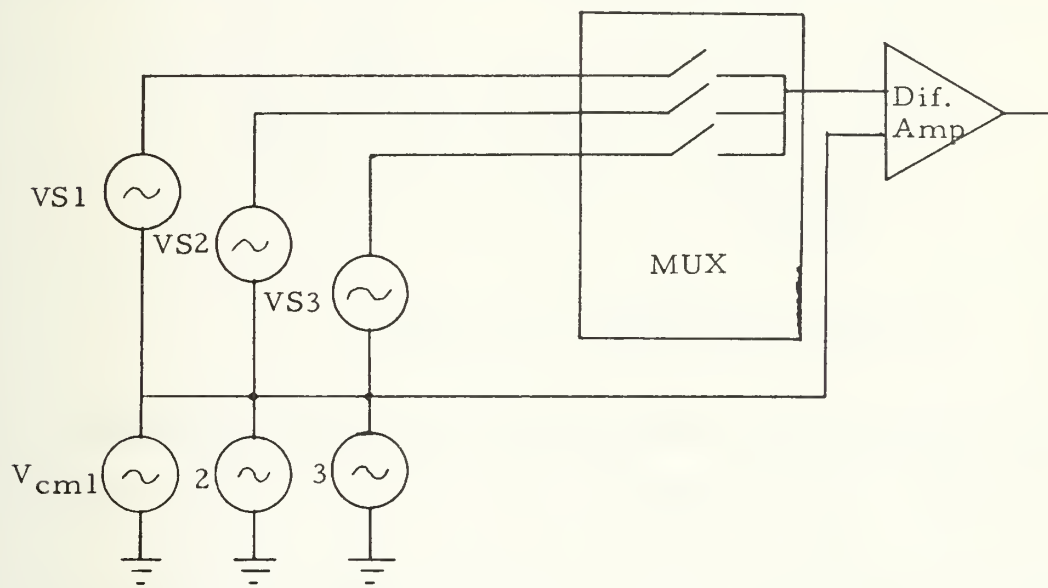


Figure 3.8 Single-ended Input Mux

To alleviate this problem, the multiplexer can be ordered in the differential mode. This is shown in Figure 3.9.

The differential amplifier can reject the common mode voltage because both the HI and LO side of the signal are switched into the amplifier. This reduces the number of channels available from 16 to 8, but the multiplexer can be expanded to 64 channels by using additional equipment.

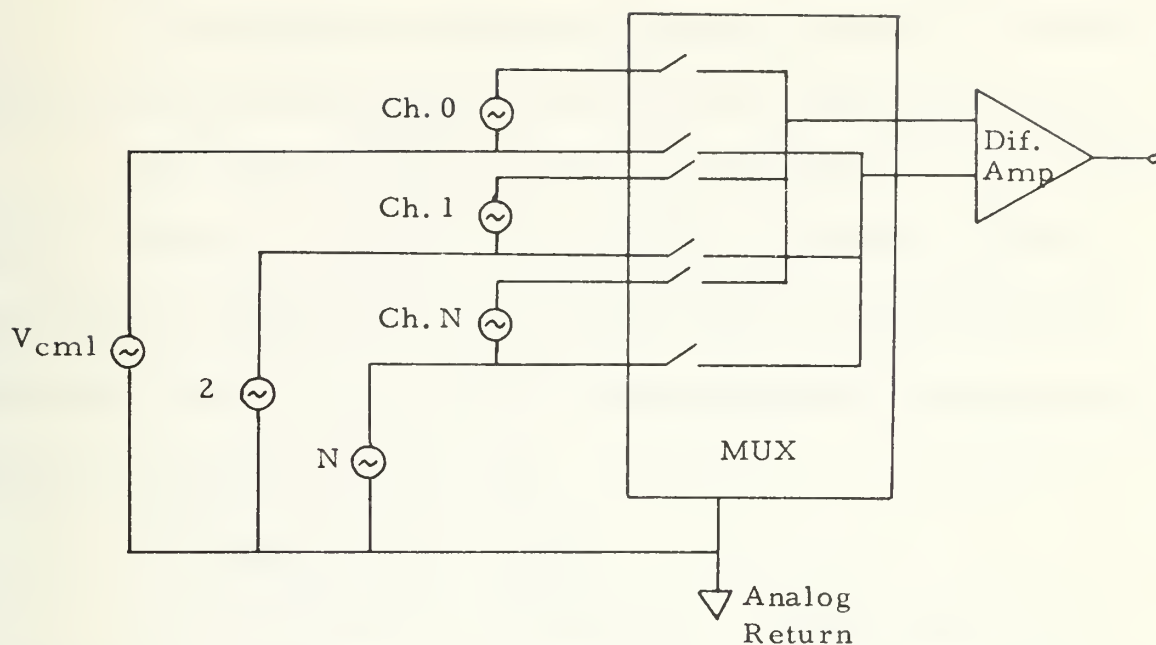


Figure 3.9 Differential Input Mux

The differential amplifier has an offset of $\pm 5\text{mV}$ which can be adjusted out when using the complete data acquisition function. The input may be ordered split from the multiplexer so that a switched gain amplifier or log amplifier can be inserted.

The multiplexer is controlled by strobes A and B. Two strobes are used so that the multiplexing operation can be inhibited if need be. Once a strobe is received, the multiplexer will do one of the following:

- a. Clear to channel 0 if Clear Enable is LO, regardless of the state of Load Enable.
- b. Advance to the next channel if Load Enable is HI and Clear Enable is HI.
- c. Select the address at A_1-A_8 if Load Enable is LO and Clear Enable is HI.

If all multiplexer channels are not used, the MUX can be reset at a predetermined address by using a NAND gate with inputs of A_1 - A_4 .

The strobe command initiates a signal which times out a period of time sufficient to allow the MUX and Sample and Hold to settle the input signal to sufficient accuracy (.01%) to allow the A/D conversion. This signal may be used to indicate when settling has occurred and may be used to start an A/D conversion.

2. The Sample and Hold Circuit

The S/H circuit is a true integrator. The settling time for the MUX and S/H is 7 μ sec. The S/H is normally in the Hold mode, but when the strobe command appears it switches to Sample. It returns to Hold after the settling time is completed. The aperture uncertainty is less than 10 nanosecs.

3. 12-Bit A/D Converter

The module uses a 12-bit successive approximation converter. The converter is triggered immediately after the S/H has settled. The data remains stored until the next conversion is initiated.

Three different codes may be obtained from the converter by use of either MSB or $\overline{\text{MSB}}$. Using $\overline{\text{MSB}}$ yields One's complement code. Using MSB provides straight binary coding for Unipolar signals and Offset Binary for Bipolar signals. Figure 3.10 shows the input voltage ranges and the respective output coding.

<u>CODE</u>	<u>10 V FULL SCALE</u>	<u>20V FULL SCALE</u>	<u>12-BIT OUTPUT CODE</u>
Binary	+9.9976 0.0000		111111111111 000000000000
Offset Binary	+4.9976 0.0000 -5.0000	+9.9951 0.0000 -10.0000	111111111111 100000000000 000000000000
1'S Complement	+4.9976 0.0000 -5.0000	+9.9951 0.0000 -10.0000	011111111111 000000000000 100000000000

Figure 3.10. DT 1620 Output Coding

4. Special Options

The DT 1620 was ordered with the following special options:

a. SG

This option allows the input to the differential amplifier to be split with the leads placed on external pins 11U, 11L, 17U and 12L. With this option, the user can insert a switched gain or log amplifier if required.

b. BR

This option allows the maximum input voltage range to be +10.24V. This gives the flexibility of adapting the converter for even binary ranges if desired. The converter may also be adjusted for a +10.00V range.

Additional information on the DT 16XX series is contained in Reference [A-1].

F. THE DT 212

Along with the DT 1620, Data Translation also produces a device used for computer control of oscilloscopes. The device has two channels of D/A conversion and numerous mode options which can be used in numerous applications including X-Y plotter and strip chart control.

Each module contains two 12-bit D/A converters, two input storage registers, two power amplifier outputs which can drive normal loads through up to 50 ft of coaxial cable, and four selectable modes for external control. A simplified schematic is shown in Figure 3.11.

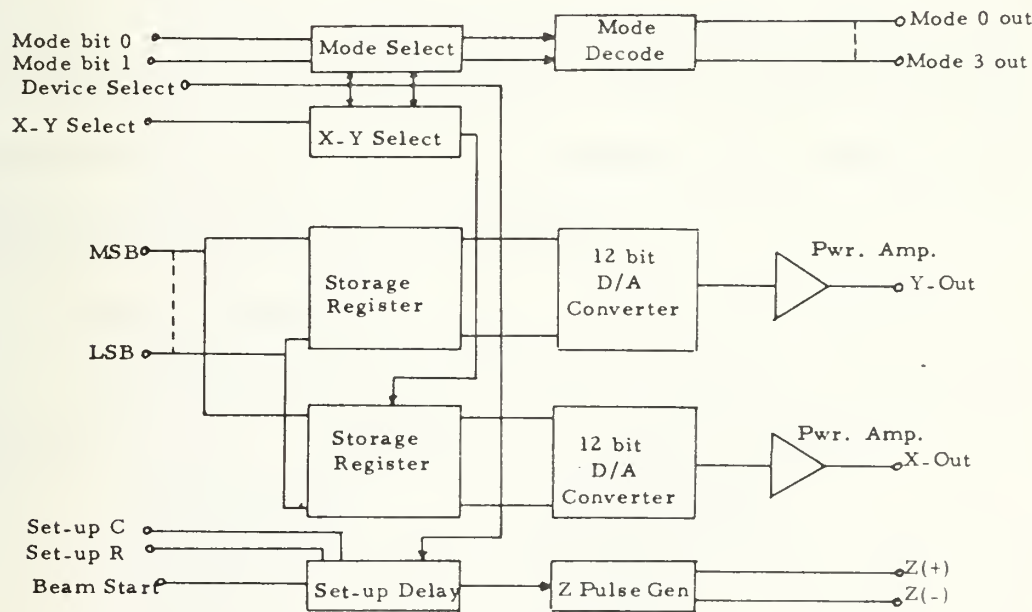


Figure 3.11 DT 212 Simplified Schematic

Referring to the schematic, the normal sequence of operation is to select either the X or the Y output channel with the X-Y select bit and then to strobe in the data which is accepted only by the storage register on the selected channel.

G. INTERFACE CONSIDERATIONS

The general interface board is designed to service one device; but to reduce the space needed for the applications, both the A/D and D/A converters were placed on one board. This is possible because only the A/D converter requires use of the DONE and BUSY flipflops. The D/A converter merely processes the data word presented to it when it is strobed and has no substantial time delay that requires the processor to wait for completion of the conversion. There are three other major interface considerations. These are device selection, implementation of the proper control signals, and setting the DONE flipflop at the end of an A/D conversion.

1. Device Selection

Referring to Figure 3.4, the device selection circuit is a six-input nand gate. The inputs are strapped as follows:

<u>Nand Input</u>	<u>Inverter Output</u>
W7/W8	U1-PIN2
W9/W10	U1-PIN10
W11/W12	U1-PIN6
W5/W6	U15-PIN8
W3/W4	U15-PIN8
W1/W2	U15-PIN12

When the nand gate is strapped in this manner, the device select signal will go HI when either code 21 (A/D converter) or code 23 (D/A converter) is presented on lines DS0-DS5. The output function is then:

$$\text{DEVICE SELECT} = \text{DS21} + \text{DS23}$$

To separate the two signals, the following circuit shown in Figure 3.12 was also placed on the board:

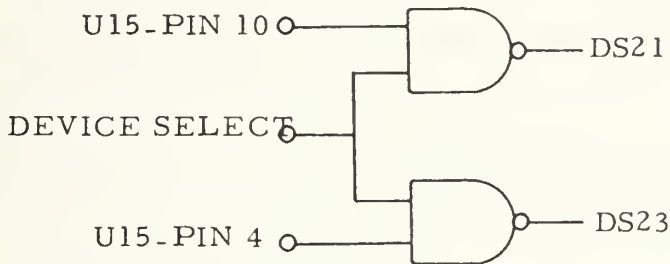


Figure 3.12 DEVICE SELECT Separation

The above circuit functions because the only difference between the two codes is the signal on DS4. When this signal is HI, and DEVICE SELECT is HI, code 21 has been called. When the signal is LO and DEVICE SELECT is HI, code 23 has been called.

2. Control Signals

There are five separate control signals that flow from processor to the interface board. They are:

- a. A pulse to reset the DONE flipflop indicating that the A/D converter has started a conversion.

- b. A pulse to actually start the A/D conversion.
- c. A pulse to load the data from the A/D converter onto the data lines.
- d. A pulse to enable the selection of the X or Y channel of the D/A converter.
- e. A pulse to enable the actual D/A conversion.

Referring to Figure 3.3, all control signals are enabled by the DEVICE SELECT signal. Since this signal is HI for the selection of either device, two possibilities exist for separate control of each device.

These are:

- 1. AND the control signal with the previously separated device selection signals
- 2. Use a different control signal for each device

Since the first alternative involves considerable additional circuitry, the second method was chosen.

a. DONE Flipflop Control

The DONE flipflop is reset by the $\overline{\text{STRT}}$ pulse. This requires that the pulse to start the A/D conversion originate from an instruction which has an "S" suffix.

b. A/D Converter Control

The $\overline{\text{DATOC}}$ signal was chosen to start the A/D conversion. The corresponding instruction is:

DOCS A, 21

where: A is the accumulator which holds
the channel address of the multi-
plexer.

c. Data Bus Control

The $\overline{\text{DATIC}}$ signal was chosen to strobe the converted data onto the data lines when the processor calls for a data word to be input to an accumulator. The corresponding instruction is:

DIC A, 21

where: A is the destination accumulator for
the converted data

d. D/A Channel Selection

The $\overline{\text{DATOB}}$ signal was chosen to enable the channel selection of the D/A converter. The corresponding instruction is:

DOB A, 23

where: A is the accumulator which contains
"0" for X channel and "1" for
Y channel

e. D/A Conversion Enable

The $\overline{\text{DATOA}}$ signal was selected to strobe the data from the data bus into the D/A converter. The corresponding instruction is:

DOA A, 23

where: A is the accumulator which con-

tains the data word to be converted.

A timing diagram for the control signals is shown in Figure 3.13. A description of each signal is given in Appendix C.

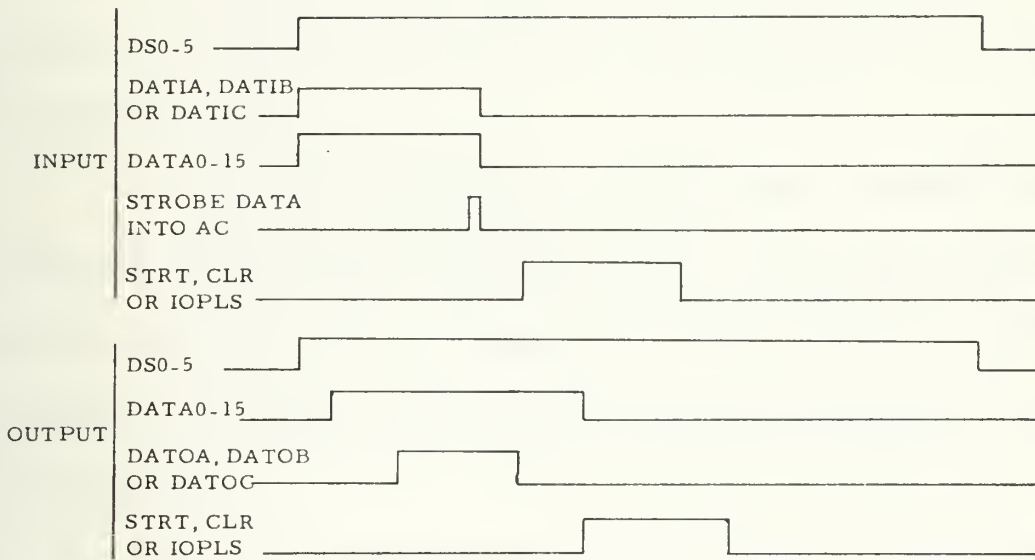


Figure 3.13 Relative Timing Diagram
For Programmed Transfers

3. Setting the DONE Flipflop

The signal used to set the DONE flipflop, the indication that the A/D has completed a conversion, is EOC (Pin27L). This signal goes high when the A/D is triggered, which is approximately 7 usec after the command to start the conversion has been given. The time delay is used by the MUX to sample a chosen signal. EOC remains high until the A/D conversion is completed and then goes low. The

point of difficulty here is the length of time that EOC remains low. In its present form, it is low when the next sample command is given. Since the START pulse is part of the sample command and occurs prior to the MUX setting (7 usec), the $\overline{\text{STRT}}$ control signal tries to reset the DONE flipflop while the EOC signal holds it set. This causes the DONE flipflop to remain set and when the processor samples the flipflop to determine the state of the device, the indication is that it is prematurely done.

To alleviate this problem, it was decided to shorten the width of the EOC signal to approximately 2 usec. This insures that the condition defined above does not occur. An N74123 Monostable Multivibrator was used as a pulse shaper. The device connection is given in Appendix D.

Another interface consideration was that the output of the other interface options is in parallel with the pins to be used for input data. These options are not needed in this application but would be required if DMA were incorporated into the board. For this reason the etched connections were opened slightly to break the electrical connection. If DMA is needed in the future, reconnection of this device could be easily performed.

Finally, the A/D system produces a 12-bit word. The data bus allows for a 16-bit word transfer. To avoid the possibility of sending erroneous information to the processor on the unused lines, the inputs to these lines were placed at digital ground.

H. CALIBRATION OF THE A/D AND THE D/A CONVERTERS

1. A/D Converter

Calibration of the A/D converter must be performed by placing the interface board on an extender board. The computer must be turned on to supply the necessary power but no computer operation is required.

The A/D converter is placed in a freerun mode by inserting a jumper between the End of Conversion (pin 27L) and Strobe A (pin 24U). In this configuration, the converter runs at its maximum throughput rate without the processor (50 kHz). Clear Enable (pin25L) is then placed at digital ground to force the continuous selection of channel 0.

A voltage of -0.0024V is placed on the channel 0 input. This is the point where MSB is just turning off and all other bits are just turning on. MSB is monitored on an oscilloscope and the offset pot (on the module nearest the computer) is adjusted until the MSB is on 50% of the time.

A voltage of -9.9976V (-10.2376V for $\pm 10.24\text{V}$ operation) is applied to the input pins of channel 0. At this point $\overline{\text{MSB}}$ is on, bits 1-10 are off, and LSB is switching between on and off. The LSB is monitored on an oscilloscope and the range pot (on the module, away from the computer) is adjusted until the LSB is on 50% of the time.

2. D/A Converter

Calibration of the D/A converter may be performed without the use of an extender board. The XO, XG, YO, YG potentiometers

are placed on the interface board to be fully accessible from the exterior of the mainframe.

Calibration Procedure:

- a) Place 000000 in AC0
- b) Place 03777 in AC1
- c) Place 62023 in location 00400
- d) Place 65023 in location 00401
- e) Execute instruction in location 00400
- f) Monitor X channel output on DVM and adjust XO
for minimum reading
- g) Execute instruction in location 00401
- h) Adjust XG for +10 volts
- i) Place 00001 in AC0
- j) Execute instruction in location 00400
- k) Monitor Y channel output on DVM and adjust YO for
minimum reading
- l) Execute instruction in location 00401
- m) Adjust YG for +10 volts.

I. SYSTEM PROGRAMMING

In Chapter II some general programs were introduced to illustrate the capabilities of the computer as it comes from the manufacturers. Consideration is next given to the modifications which must be made in order to operate it as a controller,

The first consideration is that the computer with a limited amount of core of 8K is not capable of operating on floating point numbers. Each word contains five digits and therefore some convention had to be made to convey the information as an integer. In section E of this chapter it was noted that the DT 1620 could be adjusted for either +10.00V or +10.24V operation. For the purpose of the first few programs, assume that the converter is adjusted for a +10.24V input.

The convention used was to represent +10.24 by +10240 and to remember that the last three digits are to the right of the decimal. The converter has twelve bits and therefore can represent a decimal equivalent of +2047. The approach used was to multiply this number by five and then add five to the result for positive numbers and just multiply by five for negative numbers to give a result of +10240 decimal. This introduces a maximum offset error of +00005 for voltages near zero and directly corresponds to 5 millivolts. It was felt that this amount of error should cause little trouble, but the operator should realize that the accuracy of the output will be +0.005 volts. The following program called .MAP is used to make this conversion:


```

                                ;MAPPING SR.
                                ;N IN AC1
                                ; FOR OUTPUT OF +- 10.24 VOLTS
. ENT . MAP
. NREL
. MAP: SUBO      0,0      ;CLEAR AC0
      STA      1,INT1    ;SAVE DATA
      LDA      2,MSK1    ;LOAD AC2 WITH MASK1
      AND      2,1,SNR   ;AND MSK WITH AC1 , SKIP
                                ; IF NON-ZERO
      JMP      . +6      ;JUMP IF DATA POS
      LDA      1,INT1    ;RESTORE DATA
      LDA      2,MSK2    ;LOAD MASK2 IN AC2
      ADD      2,1      ;MAKE DATA NEG
      NEG      1,1      ;FORM ABS N IN AC1
      JMP      . +6      ;JUMP N NEG
      LDA      2,OFST    ;AC2=5
      LDA      1,INT1    ;RESTORE DATA
      MUL      ;AC1=5N
      ADD      2,1      ;AC1=5N+5
      JMP      0,3      ;RETURN FOR N POS
      LDA      2,OFST    ;AC2=5
      MUL      ;AC1=5(ABS N)
      NEG      1,1      ;RESTORE N NEG
      JMP      0,3      ;RETURN FOR N NEG

OFST:  5
INT1:  0
MSK1:  4000
MSK2:  -10000
. END

```

First the input is tested to determine its sign. If it is positive, the output is $5(N)+5$. If it is negative, the absolute value is formed, the product $5(N)$ is formed, and the result is made negative. The available signed multiplication routine could have been used but since the program is fairly short, it was decided that this procedure would suffice.

Using the previous subroutine, a program was devised to test the operation of the A/D converter. The process was to measure a voltage,

map it to the proper range, convert the result to decimal and output it to the teletype. The conversion process is initiated by striking any key on the teletype. The program, called PGT1, is listed below:

```

        .ZREL
        .ENT PGT1
        .EXTN .BIND, APUT, .MAP
        .LOC 41
        APUT
BINDE:  .BIND
MAPE:   .MAP
        .NREL
PGT1:   SUBZL      0,0      ;PLACE CH1 IN AC0
        DOCS      0,21    ;LOAD ADDRESS/START
                                ;CONVERSION
        SKPDN     21      ;WAIT FOR END OF CONVERSION
        JMP      .-1      ;LOOP
        DIC      1,21    ;READ DATA TO AC1
        JSR      @MAPE   ;JUMP TO MAPPING SR
        JSR      @BINDE  ;JUMP TO OUTPUT SR
        NIOS     TTI     ;START
        SKPDN     TTI     ;SKIP IF DONE
        JMP      .-1      ;LOOP
        JMP      .-11     ;JUMP TO START
        .END PGT1

```

In the above program, Channel zero of the A/D converter is called by placing a zero in AC0 and starting the process with a DOCS 0,21 command. Code 21 is the A/D converter. The data is read into AC1, mapped, converted to decimal and then output to the teletype. The program then sits in a teletype input loop until a key is struck, at which time the process repeats.

The number conversion previously explained must also be used to convert input data that is in decimal to the format used by the A/D converter. In other words, the mapping process must be reversible.

Program .DAP was written to make this conversion:

```
                                ;DEMAPPING SR
                                ;INPUT AND OUTPUT IN AC1
      .ENT .DAP
      .NREL
.DAP:  SUBO      0,0      ;CLEAR AC0
      STA      1,DNT1    ;SAVE DATA
      LDA      2,DMSK    ;AC2=DMSK
      AND      2,1,SZR   ;AND DMSK WITH AC1, SKIP IF ZERO
      JMP      .+6       ;JUMP IF DATA NEG
      LDA      1,DNT1    ;RESTORE DATA
      LDA      2,DSET    ;AC2=5
      SUB      2,1       ;AC1=N-5
      DIV      ;AC1=(N-5)/5
      JMP      0,3       ;RETURN FOR N POS
      LDA      1,DNT1    ;RESTORE DATA
      NEG      1,1       ;FORM ABS N
      LDA      2,DSET    ;AC2=5
      DIV      ;AC2=ABS N/5
      NEG      1,1       ;RESTORE N NEG
      JMP      0,3       ;RETURN FOR N NEG

DNT1:  0
DMSK:  40000
DSET:  5
      .END
```

The above program merely performs the reverse of the operation performed by .MAP.

Using the above routine a program was devised to test the operation of the D/A converter. The procedure was to use subroutine .DBIN to fetch the number from the teletype and convert it from decimal to binary, demap to the proper range, and output it to the D/A converter. This program, called PGT2, is listed below:


```

                                ;D/A TEST
                                .ZREL
                                .ENT PGT2
                                .EXTN .DAP, .DBIN, AGET
                                .LOC 40
                                AGET
DAPE:   .DAP
DBINE:  .DBIN
                                .NREL
PGT2:   SUBO      0,0      ;CLEAR AC0
                                JSR      @DBINE ;JUMP TO INPUT ROUTINE
                                JSR      @DAPE  ;JUMP TO DEMAPPING SR
                                DOB      0,23   ;SELECT X CHANNEL
                                DOA      1,23   ;START D/A
                                JMP      .-4    ;JUMP TO START NEXT TEST

                                .END PGT2

```

In the above program, conversion is started by the DOC 2,23 command.

Code 23 is used for the D/A converter.

The next phase of testing involved placing the A/D and D/A converter together in the same program. For this and the remaining programs in this section, the A/D converter was adjusted so the max-range is +10.00V. Since the maximum range of the D/A converter is also +10.00V, no difference in bit weight will exist between input and output.

The procedure was to sample a signal and output it through the D/A converter as quickly as possible. In this way the approximate maximum through-put rate could be ascertained. The program, called SND, was written for this purpose and is listed below:


```

                                ;CONTINUOUS SAMPLING
      .ENT  SND
      .NREL
SND:  SUBO      0,0  ;CLEAR AC0
      DOB       0,23 ;SELECT X CHANNEL OF D/A
      DOAS      0,21 ;SELECT CHANNEL 1 of A/D AND START
                        ;CONVERSION
      SKPDN     21   ;SKIP IF DONE
      JMP       .-1  ;WAIT
      DIC       1,21 ;SAMPLED DATA TO AC1
      DOA       1,23 ;SAMPLED DATA TO D/A FOR OUTPUT
      JMP       .-5  ;LOOP
      .END  SND

```

Using the above program, a sinusoid was fed to the A/D input and the sample time was measured. The result was a sample width of 34 usec, which corresponds to a maximum through-put frequency of 29.4 kHz.

The final test was to take the previous program and add the capability of adjusting the time delay. Since the maximum clock frequency is 1 kHz the minimum sample time is 1 msec. A program, called SNDT, was used for this and is listed below:


```

                                ;CONTINUOUS SAMPLING WITH
                                ;ADJUSTABLE TIME RELAY
                                . ENT SNDT
                                . NREL
SNDT:  SUBO      0,0           ;CLEAR AC0
        DOB      0,23         ;SELECT CHANNEL X OF D/A
        LDA      2,DELAY      ;LOAD DELAY TO AC2
        STA      2,COUNT      ;STORE DELAY IN COUNT
        LDA      2,HZ1        ;LOAD CLOCK FREQ CODE IN AC2
        DOAS     2,RTC         ;START CLOCK FREQ=1KHZ
        SKPDN    RTC          ;WAIT FOR CLOCK PULSE
        JMP      .-1          ;LOOP
        DSZ      COUNT        ;DECREMENT COUNT, SKIP IF
                                ;ZERO
        JMP      .-4          ;LOOP
        DOCS     0,21         ;START A/D
        SKPDN    21           ;WAIT FOR CONVERSION
        JMP      .-1          ;LOOP
        DIC      1,21         ;LOAD CONVERTED DATA TO AC1
        DOA      1,23         ;LOAD AND START D/A
        JMP      .-15         ;RECURSE

DELAY: 1
COUNT: 0
HZ1:   3
      . END SNDT

```

In the above program the procedure for establishing the time delay is the same as was used for the CLOCK TEST. A delay of one will give a 1-msec sample rate and may be adjusted by changing either the values of DELAY or HZ1 (the clock frequency) or both.

IV. SYSTEM OPERATION TUTORIAL

A. INTRODUCTION

This chapter is designed to be used as a tutorial for one to become familiar with the operation of the Nova computer. The information is given in a concise form so that the chapter may be used as a quick reference.

First the turn-on procedure for each device in the system is discussed. Next, a step-by-step procedure is given for initiating each of the software routines discussed in Chapter II along with a sample programming sequence for this utilization.

Finally, a programmed teaching sequence is offered to help the user to become proficient in using the computer in the most expedient manner. The procedure is designed to be initially used in order, but after the user has gone through the sequence once, each section may be used as a refresher on a particular utility routine.

B. MACHINE OPERATION

1. Turn On

a. Computer - The power switch is on the left side of the front panel and has three positions: OFF, ON, LOCK. The lock position inhibits control by the operator from the front panel. The ON position allows front panel control. The ON position is normally used.

b. Cassette tape deck - The power switch is on the upper right-hand corner of the deck and has three positions: OFF, REMOTE, LOCAL. In REMOTE, power is controlled by the computer power switch. In LOCAL the power is applied directly. The REMOTE position should be used as it allows additional cooling by the fan after power shutoff.

WARNING: When the tape deck is turned on, voltage spikes are present. Tapes should not be mounted until after turn on to avoid possible destruction of data.

c. Teletype - The power switch is located on the front, right-hand side of the machine and has three positions: OFF, ON-LINE, LOCAL. LOCAL allows the teletype to be used by itself, such as typing the contents of a paper tape. In this position it is not connected to the computer. The normal position is ON-LINE.

d. Paper Tape Reader - The power switch is on the front of the reader and has three positions: START, STOP, FREE. The FREE position allows the tape to move freely through the sprocket to check alignment. The START position starts the tape if the teletype is in LOCAL mode and places the reader under computer control if the teletype is in ON-LINE.

e. Paper Tape Punch - ON and OFF are buttons on the panel. When the ON button is depressed, data that is received by the teletype from the computer will be punched on a paper tape.

2. Loading the image core loader/writer

a. Set 100034 on the data switches. This indicates a high

speed device with code 34 which is the tape drive 0.

b. Place the master tape on tape drive 0 and rewind.

c. Momentarily depress STOP and then RESET.

d. Momentarily push the Program-load switch up.

e. When computer stops, put 0 data switch down and

momentarily depress CONTINUE.

f. Computer will respond with "#" when load is complete.

3. Loading the CLI

a. Load the image core loader/writer.

b. After "#" response, type 1 and carriage return.

c. Computer will respond with "R" when load is complete.

4. Restarting the image core loader

a. Set 017777 in the data switches.

b. Momentarily push RESET and then START.

c. Computer will respond with "#" when load is complete.

5. Loading the Text Editor

a. Load CLI, type EDIT and carriage return,
OR

b. Load image core loader, type 2 and carriage return.

6. Loading the Assembler

a. Load CLI, type ASM and carriage return,
OR

b. Load image core loader, type 3 and carriage return.

7. Loading the Relocatable Loader

- a. Load CLI, type RLDR and carriage return,
OR
- b. Load image core loader, type 4 and carriage return.

8. Sample Procedure

- a. Turn on machine.
- b. Load image core loader/writer.
- c. Load text editor.
- d. Build/correct program and place it on tape.
- e. Load assembler.
- f. Assemble program with binary output to a second tape.
- g. If no diagnostics, load relocatable loader; otherwise

go to d above.

- h. Load binary program and establish a save file on a third tape.

- i. Execute program.

The program is now in three forms on three different tapes. The save tape is all that is needed to use the program at a future time. It contains the main program and all subroutines on one file and is in relocatable binary form. The source tape should be saved in case the save tape is damaged. The binary tape may be saved as the user wishes.

C. SYSTEM FAMILIARIZATION

1. Programming

- a. Read Reference [D-3]
- b. Review programs given in Chapters II and III

2. Self-Loading Bootstrap and Image Core Loader/Writer

- a. Read Reference [D-3], page 5-1 and 5-2
- b. Read Chapter II, Section E-1

3. Command Line Interpreter

- a. Read Reference [D-3] page 5-3 through 5-4
- b. Read Chapter II, Section E-2

4. Text Editor

- a. Read Reference [D-4], exclude Stand-alone Operation

Page 3-1

- b. Read Chapter II, Section E-3
- c. Exercise for using the Text Editor

1. Turn on equipment (Section B-1 of this chapter)

2. Load the image core loader/writer (Section B-2

this chapter)

3. Load the Command Line Interpreter (Section B-3

this chapter)

4. Type "EDIT (carriage return)"

The editor will respond with "OK" when it is loaded

5. Type the following message:

I(TAB) THE NOVA COMPUTER IS A 16 BIT
(carriage return) WORD LENGTH MACHINE WITH
FOUR (carriage return) WORKING ACCUMULATORS.
(carriage return) \$\$

6. Mount a scratch tape on cassette drive 1
7. Check the message by typing B\$T\$\$
8. Place the message on the scratch tape by typing
GWCT1:0\$B\$P\$GC\$\$
9. Clear the text editor buffer by typing 999K\$\$

The buffer may be checked by typing B\$T\$\$;
No message will be printed.
10. Read the message back into the buffer by typing
GRCT1:0\$Y\$\$
11. Type out the buffer by typing T\$\$
12. Use the commands listed in Reference [D-4] to change

the message and type out either a line or the entire buffer to check the result.

13. When you are familiar with the commands type H\$\$
to return to the loader mode.

It is important to realize that once the text editor mode is left, the buffer is destroyed, so after writing or altering a program, remember to save it on a scratch tape.

5. Assembler

- a. Read Reference [D-5], exclude pages 38-40
- b. Read Reference [D-6], exclude Appendix A

c. Read Chapter II, Section E-4

d. Exercise for using the Assembler

1. Perform steps 1-4 of Text Editor exercise

2. Type, on the teletype, a copy of the program entitled

PGM in Chapter II and place it on a scratch tape on cassette drive 1,

file 0

3. Place 17777 in the data switches

4. Depress RESET and then START

This will place the computer in the loader mode

5. Type "3 (carriage return)"

This will load the assembler which will respond with

"ASM"

6. Remove the master tape and place another scratch

tape on drive 0

7. Type "ASM CT1:0 CT0:0/B \$TT0/L (carriage

return)"

a. The computer will output a listing of the assembled program and produce a binary output on tape drive 0, file 0

b. Save this tape for the Relocatable Loader exercise

8. Repeat steps 3 through 7 but use a program of your own.

The first attempt at programming should produce some diagnostic errors. Correct these by using the Text Editor and Assembler.

6. Relocatable Loader

a. Read Reference [D-7], excluding Section 1

b. Read Chapter II, Section E-5

c. Exercise for using the Relocatable Loader

1. Mount the master tape on drive 0 and the binary tape of PGM produced in the Assembler exercise on tape drive 1

2. Load the Command Line Interpreter

3. Remove the master tape and replace it with the tape containing the assembled .BIND and APUT subroutines

4. Type "XFER CT0:0 CT1:1 (carriage return)"

This places the assembled .BIND subroutine on drive 1, file 1

5. Type "XFER CT0:4 CT1:2 (carriage return)"

This places the assembled AGET subroutine on drive 1, file 2

6. Remove the subroutine tape from drive 0 and replace it with the master tape

7. Type "RLDR (carriage return)"

This places the computer in the Relocatable Loader mode. The computer will prompt "RLDR" when the load is complete. .

8. Remove the master tape and replace it with another scratch tape

9. Type "RLDR CT1:0 CT1:1 CT1:2 CT0:0/S \$TTO/L"

The computer will type the listing on the teletype, load the program starting in location 00440, and produce a save file consisting of the main program and the subroutines

10. To start the program set 00440 in the data switches

and depress EXAMINE and CONTINUE

The sum of the number X and Y (i. e., 11) will be output on the teletype

11. Change the numbers X and Y

- a. Set the location of X in the data switches and depress EXAMINE
- b. Set the binary equivalent of the new numbers on the data switches and depress DEPOSIT
- c. Repeat (a) and (b) for Y and restart the program as in (10)

12. The save file is all that is needed to initiate the pro-

gram at a future time. To investigate this, perform the following steps:

- a. Load the Command Line Interpreter
- b. Place the save tape on drive 1
- c. Type "CT1:0 (carriage return)"

The program will be loaded and can be started as before

13. Note: If the main program (PGM) had been the last one

placed on the save file while in the relocatable loader mode, the program would start automatically when loaded.

This may be done by reentering the relocatable loader mode and reconfiguring the save tape with the subroutines loaded first and the main program loaded last.

V. EXAMPLE CONTROL PROBLEMS

A. INTRODUCTION

To demonstrate the use of the Nova as a digital controller, a continuous plant was simulated on the Miniatic analog computer. The first concept to be demonstrated was that in any sampled data system the step response tends to show increased overshoot and settling time as the sampling period becomes large. To show this the system in Figure 5.1 was connected and the program called SNDT was used to vary the sampling rate.

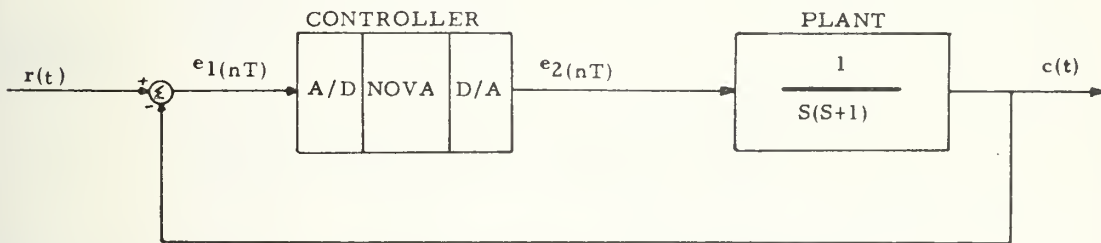


Figure 5.1 Example Control System

Figure 5.2 shows a comparison of the system step response for sample times of 1 sec and 2 sec with that of the continuous system. As expected this figure shows that care must be taken when applying digital control to a stable continuous system.

B. PROGRAM DEVELOPMENT

To perform a more complex control of the system a general purpose routine was developed for the computer. This routine must perform three distinct functions: 1) synchronize the digital and analog computers in time, 2) solve a recursion equation, and 3) control the sample time.

1. Computer Synchronization

The analog computer used produces a 5-volt pulse for the duration of its operation. This pulse is sampled by the program so that the digital and analog starts coincide.

2. Generation of the Recursion Equation

In general, the Z transfer function of the controller for the system is,

$$D(Z) = \frac{E_2(Z)}{E_1(Z)} = \frac{C_2 + C_3 Z^{-1}}{1 + C_1 Z^{-1}}$$

This corresponds to the time domain representation:

$$e_2(K) = -C_1 e_2(K-1) + C_2 e_1(K) + C_3 e_1(K-1)$$

where: $e_2(K)$ is the generated control signal

$e_2(K-1)$ is the control signal generated at the previous sample time

$e_1(k)$ is the sampled error signal

$e_1(K-1)$ is the error signal sampled at the previous sample time

C_1, C_2, C_3 are constants which govern the operation of the controller.

3. Control of the Sample Time

The sample time is controlled in much the same manner as was done in the clock test and in program SNDT. One added consideration is the time required for computation. Appendix G contains the times required for each type of instruction. These were added to the sample time required to give the total sample time. The constant DELAY was then adjusted to compensate for this generated error. The program designed for the demonstration is given below.

```
.ENT SDSM
.EXTN .MPY,.DIV
.MPYE: .MPY
.DIVE: .DIV
.NREL
SDSM: SUBZL      0,0      ;AC0=1
      SUBO       1,1      ;AC1=0
      STA        1,E1K    ;
      STA        1,E2K1   ;
      STA        1,E1K1   ;CLEAR DATA
      STA        1,GE2K1  ;
      STA        1,GE1K   ;
      DOB        1,23     ;SELECT CHANNEL X
      DOA        1,23     ;CLEAR D/A
      LDA        2,MSK1   ;AC2=MSK1
      DOCS       0,21     ;SAMPLE CHANNEL 1
      SKPDN      21       ;WAIT UNTIL DONE
      JMP        .-1
      DIC        1,21     ;SAMPLED DATA TO AC1
      AND        2,1,SNR  ;SKIP IF A 2.5 VOLTS
      JMP        .-5      ;RECURSE
      SUBO       0,0      ;CLEAR AC0
      DOCS       0,21     ;SAMPLE CHANNEL 0
      SKPDN      21       ;WAIT UNTIL DONE
      JMP        .-1
      LDA        1,GAIN3   ;AC1=GAIN3
      LDA        2,E2K1    ;AC2=E2(K-1)
      JSR        @.MPYE    ;AC0-1=GAIN3*E2(K-1)
      LDA        2,HND     ;
      JSR        @.DIVE    ;NORMALIZE
```


STA	1, GE2K1	;SAVE AC1
DIC	2, 21	;AC2=E1(K)
LDA	0, MSK2	;AC0=MSK2
SUB	2, 0, SNR	;CHECK DATA FOR 1'S COMP
		; ZERO
SUBO	2, 2	;CHANGE TO 2'S COMP ZERO
LDA	0, MSK3	;AC0=MSK3
AND	2, 0, SNR	;CHECK FOR 12 BIT NEGATIVE
		; DATA
JMP	. +3	;JUMP IF DATA POSITIVE
LDA	0, MSK4	;AC0=MSK4
ADD	0, 2	;CHANGE 12 BIT NEG DATA TO
		; 16 BIT NEG DATA
SUBO	0, 0	;CLEAR AC0
STA	2, E1K	;SAVE E1(K)
LDA	1, GAIN1	;AC1=GAIN1
JSR	@. MPYE	;AC0-1=GAIN1*E1(K)
LDA	2, HND	;
JSR	@. DIVE	;NORMALIZE
SUBO	0, 0	;CLEAR AC0
STA	1, GE1K	;SAVE DATA
LDA	2, E1K1	;AC2=E1(K-1)
LDA	1, GAIN2	;AC1=GAIN2
JSR	@. MPYE	;AC0-1=GAIN2*E1(K-1)
LDA	2, HND	;
JSR	@. DIVE	;NORMALIZE
LDA	2, GE2K1	
ADD	2, 1	
LDA	2, GE1K	
ADD	2, 1	;AC1=GAIN1*E1(K)+GAIN2*E1(K-1)
		; +GAIN3*E2(K-1)
STA	1, E2K1	;E2(K) BECOMES E2(K-1)
DOA	1, 23	;OUTPUT E2(K)
LDA	1, E1K	;AC1=E1(K)
STA	1, E1K1	;E1(K) BECOMES E1(K-1)
LDA	2, DELAY	;
STA	2, COUNT	;
LDA	2, HZ1	;
DOAS	2, RTC	;TIMER
SKPDN	RTC	;
JMP	. -1	;
DSZ	COUNT	;
JMP	. -4	;
JMP	. -60	;RESTART


```

MSK1:    3400
MSK2:    7777
MSK3:    4000
MSK4:    170000
DELAY:   1740
HZ1:     3
COUNT:  0
HND:     144
E1K:     0
E2K1:    0
E1K1:    0
GE2K1:   0
GE1K:    0
GAIN1:   420
GAIN2:   -144
GAIN3:   -110
        .END SDSM

```

First, all save addresses for intermediately generated data are cleared and the output of the D/A converter is set to zero volts. Then the signal on channel 1 is sampled and tested to see if it is greater than 2.5-volts. This signal is the control signal generated by the analog computer previously discussed. If the signal is less than 2.5-volts, it is sampled and tested again. This process continues until the 5-volt pulse is present at which time the problem starts.

After the program starts, channel 0 is sampled and becomes $E_1(K)$. In generating $E_2(K)$, the following programming points should be noted:

1. When a negative voltage is sampled, the MSB of the A/D converter is a 1. But when this digital word is read into the computer, the MSB becomes Bit 4 of the computer word which is recognized as a positive number. To compensate for this problem, the MSB is tested

and if it is 1, Bits 0-3 of the computer word are also set to 1. The computer recognizes this as a proper negative number. This was not done when the start pulse was sampled because a prior measurement indicated that this signal was never negative.

2. The A/D converter is a 1's complement device and the computer is a 2's complement machine. Since a zero for the A/D converter may be 7777 or 0000, the condition may exist where all bits in the computer word are 1's. This is an improper condition for the computer and therefore if 7777 is input to the processor, it is immediately set to 0000.

3. The gains are floating point numbers accurate to two decimal places. Since only fixed point numbers can be used, the gain multiplied by 100 is used and the result is divided by 100. The remainder of the division in ACØ is discarded.

4. The multiplication and division instructions may involve negative numbers so the manufacturer supplied signed number routines (.MPY and .DIV) were used. These routines were modified by replacing the JSR instruction within each of them by the MUL and DIV instructions. These routines were designed for machines which do not have hard-wired multiply and divide instructions but with the above modifications, the routines performed satisfactorily.

5. After a division is performed the remainder resides in ACØ. This must be cleared before any following multiplication or an error will result.

After $e_2(K)$ is generated both $e_2(K)$ and $e_1(K)$ are shifted in time. That is, they become $e_2(K-1)$ and $e_1(K-10)$ respectively and are used in the next sample period.

Finally the timer is used to generate the required delay. In this case $T=1$ second and the clock frequency is 1kHz. Therefore, DELAY should be 1000 (1750₈). Notice that 1740 was used. This is because 8 msec of compensation was needed when the instruction execution time of the program and subroutines was calculated.

With this general program written, two types of problems were to be solved. These were 1) minimum prototype design and 2) ripple free (dead beat) design. The plant to be controlled was the same as was previously used. The continuous system equations are:

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} e_2$$

$$c(t)=[01] \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

The difference equations which characterize the system at the sampling instants are:

$$\underline{x}(K) = \phi \underline{x}(K-1) + \underline{A} \underline{u}(K-1)$$

where

$$\phi = e^{\underline{A}T} = \begin{bmatrix} e^{-T} & 0 \\ 1-e^{-T} & 1 \end{bmatrix}$$

$$\underline{A} = \int_0^T e^{\underline{A}(T-\tau)} \underline{B} d\tau = \begin{bmatrix} 1-e^{-T} \\ T+1-e^{-T} \end{bmatrix}$$

For $T=1.0$ sec we have

$$\phi(1) = \begin{bmatrix} 0.368 & 0 \\ 0.632 & 1 \end{bmatrix}$$

$$\Delta(1) = \begin{bmatrix} 0.632 \\ 0.368 \end{bmatrix}$$

The system difference equation is

$$\begin{bmatrix} X_1(K) \\ X_2(K) \end{bmatrix} = \begin{bmatrix} 0.368 & 0 \\ 0.632 & 1 \end{bmatrix} \begin{bmatrix} X_1(K-1) \\ X_2(K-1) \end{bmatrix} + \begin{bmatrix} 0.632 \\ 0.368 \end{bmatrix} e_2(K-1) \quad (5-1)$$

$$c(K) = X_2(K)$$

C. MINIMUM PROTOTYPE DESIGN

The objective of this design is to force the system response to the final value in as short a time as possible. From theory, [Ref. C-1], this specific system can reach the final value in one sample time but will have a great deal of overshoot. The system will pass through the final value at each sample time. Assuming zero initial conditions and a constant input R , the design procedure is as follows:

At $K = 1$,

$$X_1(1) = 0.632 e_2(0)$$

$$X_2(1) = 0.368 e_2(0)$$

to have $X_2(1) = R$,

$$e_2(0) = \frac{R}{0.368} = 2.72R$$

$$X_1(1) = (0.632) 2.72R = 1.74R$$

At $K = 2$,

$$\begin{bmatrix} X_1(2) \\ X_2(2) \end{bmatrix} = \begin{bmatrix} 0.368 & 0 \\ 0.632 & 1 \end{bmatrix} \begin{bmatrix} 1.74R \\ R \end{bmatrix} + \begin{bmatrix} 0.632 \\ 0.368 \end{bmatrix} e_2(1)$$

$$X_2(2) = (0.632) (1.74R) + R + 0.368 e_2(1) = R$$

$$e_2(1) = -2.95R$$

Continuing in this manner we find that

$$E_2(Z) = 2.72R - 2.95RZ^{-1} + 2.12RZ^{-2} \dots \quad (5-2)$$

now,

$$E_2(Z) = D(Z) E_1(Z)$$

To find $E_1(Z)$ we note that

$$e_1(t) = r(t) - c(t)$$

and for $t = KT$

$$e_1(K) = r(K) - c(K)$$

For $K = 0$

$$e_1(0) = R - 0 = R$$

and for $K > 0$

$$e_1(K) = 0$$

therefore,

$$E_1(Z) = R \quad (5-3)$$

substituting (5-3) into (5-2) yields

$$D(Z) = 2.72 - 2.95Z^{-1} + 2.12Z^{-2} \dots$$

and in closed form:

$$D(Z) = \frac{2.72 - Z^{-1}}{1 + 0.717Z^{-1}} = \frac{E_2(Z)}{E_1(Z)}$$

which yields the recursion equation

$$e_2(K) = 2.72 e_1(K) - 1.0 e_1(K-1) - 0.717 e_2(K-1)$$

therefore,

$$\text{GAIN } 1 = 272 = 420_8$$

$$\text{GAIN } 2 = -100 = 144_8$$

$$\text{GAIN } 3 = -72 = -110_8$$

Figure 5.3 shows the step response of the system and the computed $E_2(K)$ for the above gains. The system passes through the final value at the beginning of each sample period beginning at $K = 1$.

D. RIPPLE-FREE (DEAD-BEAT) RESPONSE

The objective of this design is to force the system response to the final value as quickly as possible and have no overshoot. Again from theory, this system can satisfy both conditions in two sample periods. Assuming zero initial conditions and an input of R , the design procedure is as follows:

The system response after two sample periods is:

$$\underline{X}(2) = \underline{A}^2(T)\underline{X}(0) + \underline{A}(T)\underline{B}(T)e_2(0) + \underline{B}(T)e_2(1)$$

and

$$X(2) = \begin{bmatrix} X_1(2) \\ X_2(2) \end{bmatrix} = \begin{bmatrix} 0 \\ R \end{bmatrix}$$

therefore,

$$\begin{bmatrix} 0 \\ R \end{bmatrix} = \begin{bmatrix} 0.368 \\ 0.632 \end{bmatrix} e_2(0) + \begin{bmatrix} 0.632 \\ 0.368 \end{bmatrix} e_2(1)$$

which yields

$$e_2(0) = 1.58R$$

$$e_2(1) = 0.58R$$

placing this in a Z transform format

$$E_2(Z) = R(1.58 - 0.58Z^{-1}) = E_1(Z)D(Z)$$

Again we need to solve for $E_1(Z)$

$$e_1(0)=R$$

and

$$e_1(1)=R-C(1)=R-X_2(1) \quad (5-4)$$

Solving equation (5-1) for $X_2(1)$ yields

$$\begin{aligned} X_2(1) &= (0.368)e_2(0) \\ &= (0.368)(1.58R) \\ &= 0.582R \end{aligned}$$

Substituting into equation (5-4) we have

$$e_1(1)=R-0.582R=0.418R$$

Therefore,

$$D(Z) = \frac{E_2(Z)}{E_1(Z)} = \frac{1.58-0.58Z^{-1}}{1+0.418Z^{-1}}$$

which yields the recursion equation:

$$e_2(K) = 1.58e_1(K)-0.58e_1(K-1)-0.42e_2(K-1)$$

therefore,

$$\text{GAIN 1} = 158 = 236_8$$

$$\text{GAIN 2} = -58 = -72_8$$

$$\text{GAIN 3} = -42 = -52_8$$

Figure 5.4 shows the step response of the system and the computed $e_2(K)$ for the above gains. The system reaches the final value in two sampling times as expected after which no further control is required.

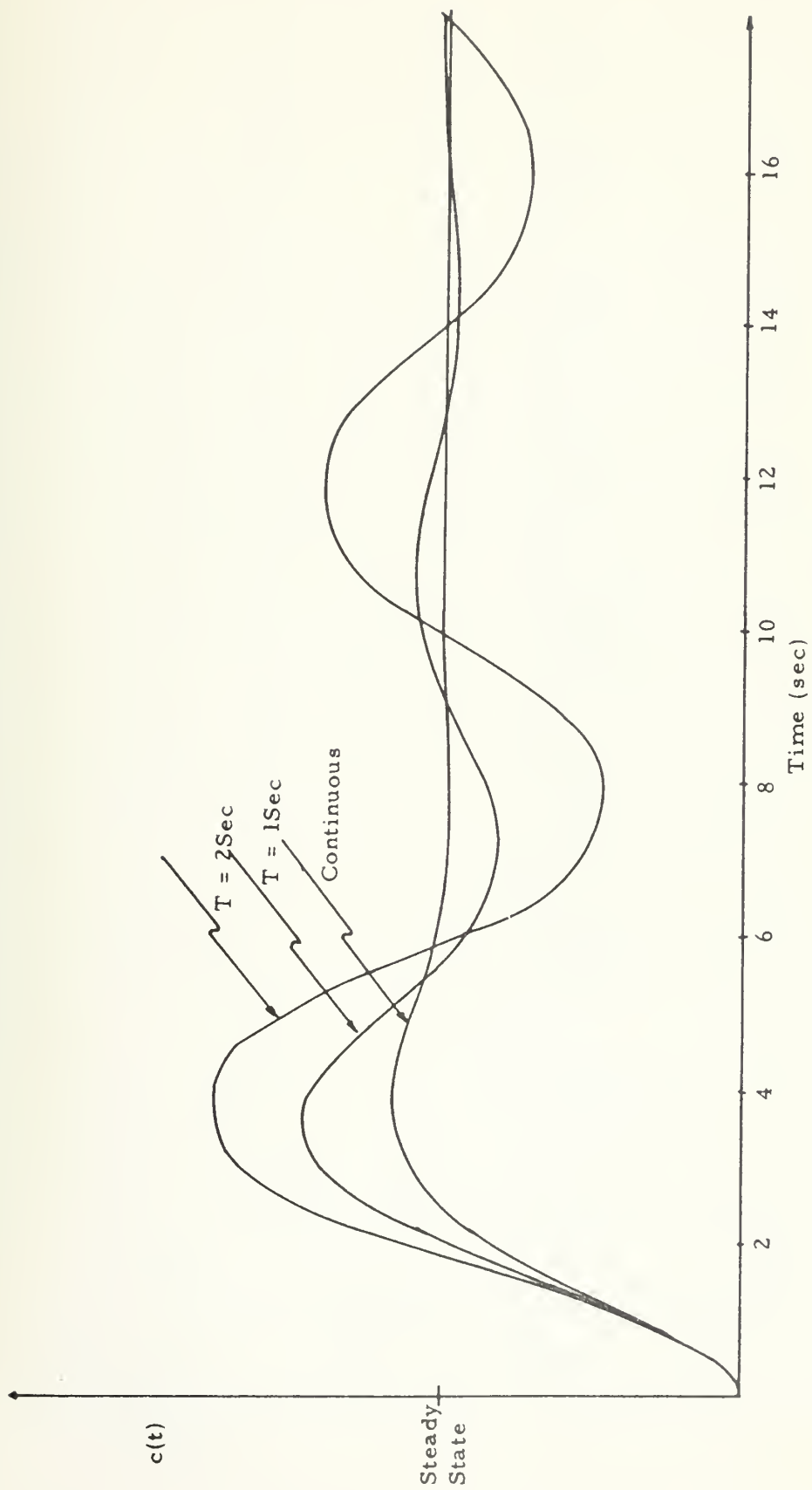


Figure 5.2 Comparison of Continuous and Sampled System

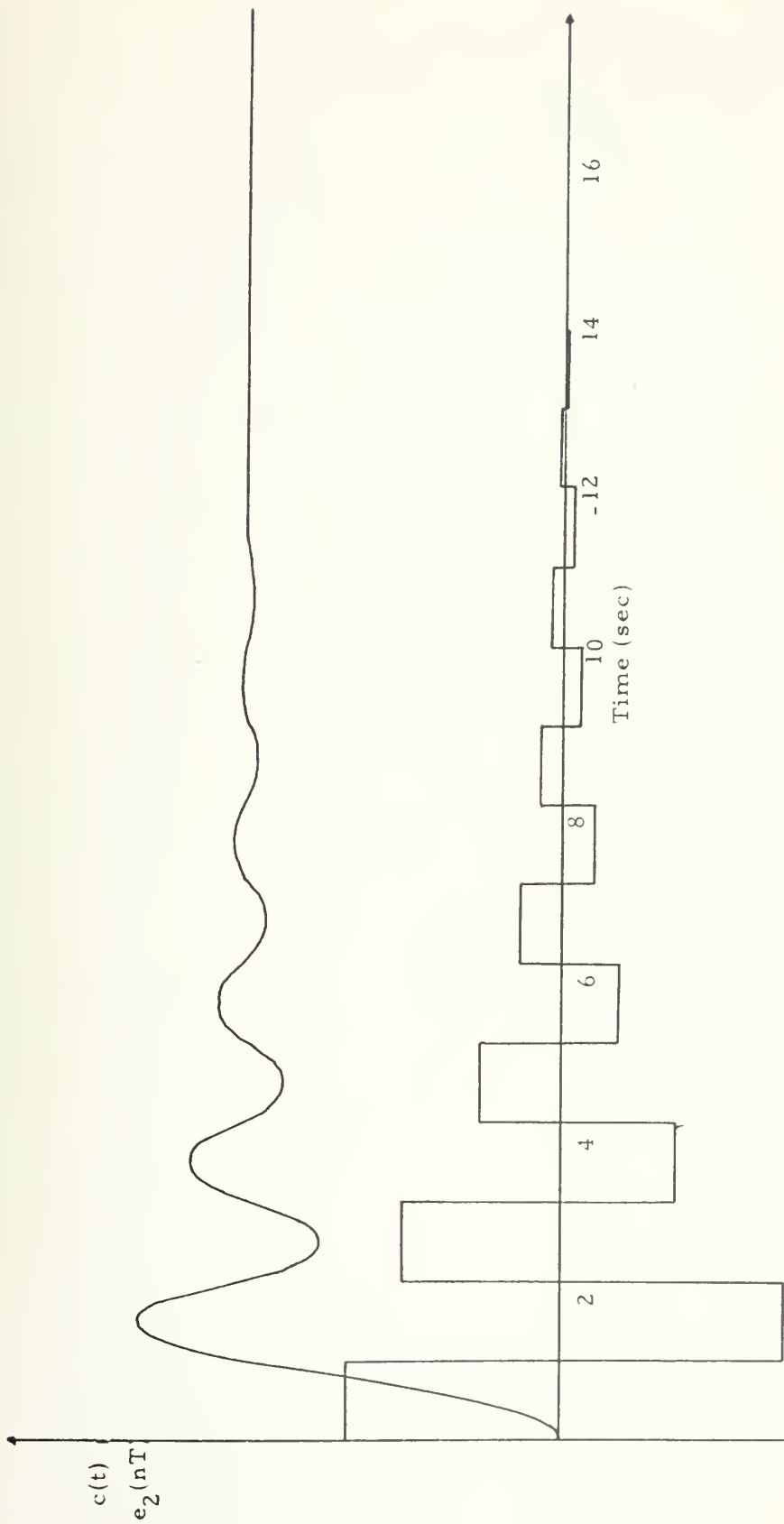


Figure 5.3 Minimum Prototype Response

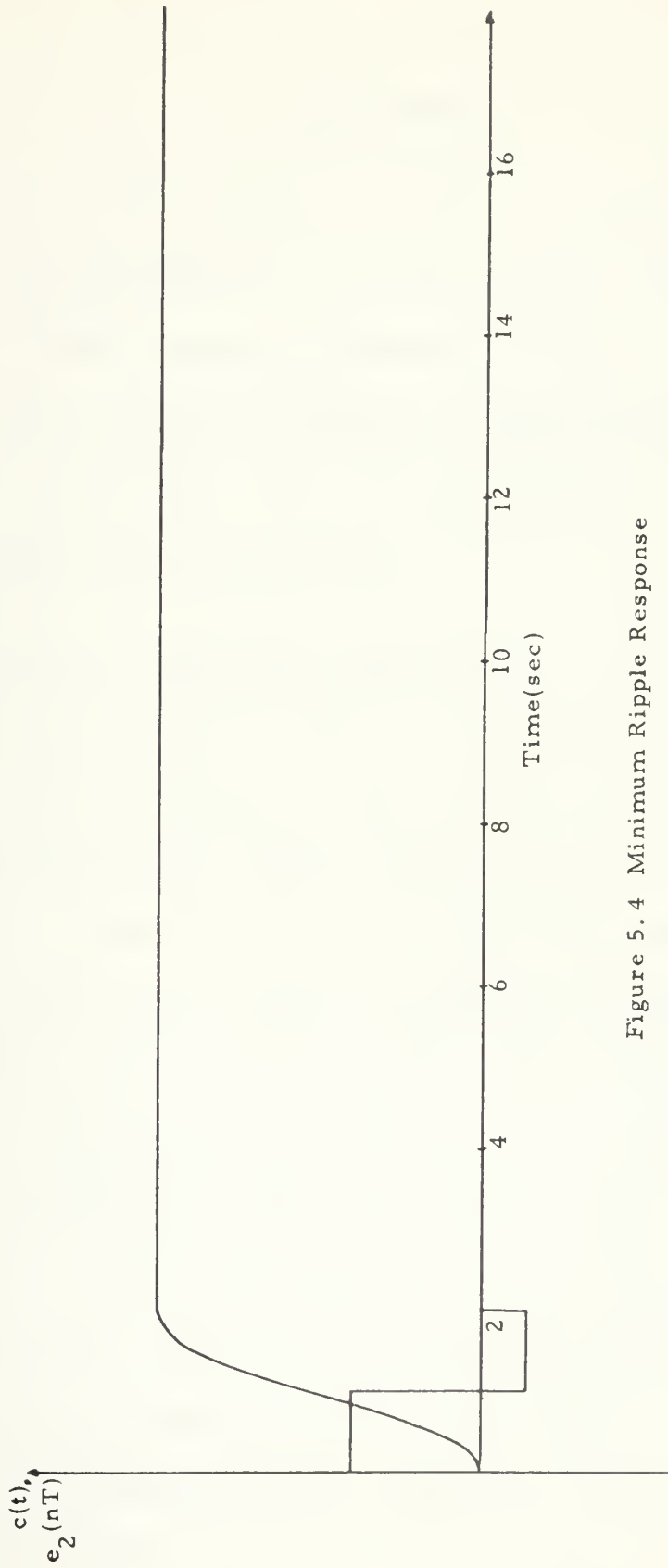


Figure 5.4 Minimum Ripple Response

VI. CONCLUSION

The substance of this project was to design, build and demonstrate a basic digital control system using a general purpose digital mini-computer as the main computational element. To this end, the project has been completed, but the possibilities for expansion and refinement of the system are numerous. Before discussing some of these an evaluation of the system elements is in order.

A. EVALUATION OF THE NOVA COMPUTER

The computer is adequate as far as a computational machine but is not oriented to user operation. That is, much of the time required to set up a control problem is spent in learning the operation of the computer, the tape filing system, and the mnemonic language. Unfortunately, the 8K of core available is insufficient to support a resident operating system and a compiler for a higher level language. The infrequent user, although experienced in large general purpose computers and having programmed in a higher level language such as FORTRAN or BASIC, must be constantly aware of the functional characteristics of the operational hardware. This detracts from the essence of the control problem by introducing many operational details which can only be learned after acquiring many hours of experience. This is the major weakness of the system.

B. EVALUATION OF THE A/D CONVERSION EQUIPMENT

The DT1620 performed extremely satisfactorily. Manufacturer support in the way of reference material supplied and response to queries was exceptional. The module represents a high quality, precision device with numerous options so that it may be applied to almost any conceivable data acquisition problem. The speed of the DT1620 is not acceptable for the high resolution requirements of communication equipment, but the DT16XX series can have a throughput rate as high as 100KHZ. This coupled with a DMA system where the memory access time is approximately 2usec would realize an overall maximum sampling rate of 83.3KHZ which is capable of providing high resolution sampling of any real system. The DT1620 when coupled with DMA will provide a maximum sampling rate of 45.5KHZ which is adequate for control systems.

C. EVALUATION OF THE D/A EQUIPMENT

The DT212, although not specifically designed for the purpose used, allows for a great deal of flexibility. The additional circuits which would have normally been used for oscilloscope control can be used as system start/stop pulses if required and if display of stored data is needed, the module can control most display devices.

D. EVALUATION OF THE SYSTEM

The overall system operated as expected. The interface of components was fairly direct and did not require a great deal of additional

logic to be added to that which was already present on the general interface board. The major deficiency of the system at this time is the lack of computer core and the resulting problems which were previously discussed.

E. RECOMMENDATIONS FOR EXPANSION

1. To improve the system capability an additional 24K of core is advised. This would give a total of 32K which is the minimum for incorporating a resident operating system and FORTRAN compiler. For a fully capable system the installation of a disk system is advised. This would allow the user to operate in FORTRAN, use floating point routines, and eliminate much of the hands on operation.

2. The final recommendation is to improve the software of the operating system. This includes the following specific items:

a. Reprogram the use of the real time clock so that it is incorporated into the computer interrupt system. This will allow the computer to process instructions during the operation of the sample timer and eliminate the need for consideration of the error in sampling time caused by the instruction execution times.

b. Design a software operating system to facilitate operator interaction with the system. This would include changing the sampling time, changing control law coefficients, and changing the form of recursion equation without complete reprogramming of the problem.

APPENDIX A

INSTRUCTION MNEMONICS

ADC	102000	Add the complement of ACS to ACD; use Carry as base for carry bit.
ADCC	102060	Add the complement of ACS to ACD; use complement of Carry as base for carry bit.
ADCCL	102160	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; rotate left.
ADCCR	102260	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; rotate right.
ADCCS	102360	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; swap halves of result.
ADCL	102100	Add the complement of ACS to ACD; use Carry as base for carry bit; rotate left.
ADCO	102040	Add the complement of ACS to ACD; use 1 as base for carry bit.
ADCOL	102140	Add the complement of ACS to ACD; use 1 as base for carry bit; rotate left.
ADCOR	102240	Add the complement of ACS to ACD; use 1 as base for carry bit; rotate right.
ADCOS	102340	Add the complement of ACS to ACD; use 1 as base for carry bit; swap halves of result.
ADCR	102200	Add the complement of ACS to ACD; use Carry as base for carry bit; rotate right.
ADCS	102300	Add the complement of ACS to ACD; use Carry as base for carry bit; swap halves of result.
ADCZ	102020	Add the complement of ACS to ACD; use 0 as base for carry bit.
ADCZL	102120	Add the complement of ACS to ACD; use 0 as base for carry bit; rotate left.

ADCZR	102220	Add the complement of ACS to ACD; use 0 as base for carry bit; rotate right.
ADCZS	102320	Add the complement of ACS to ACD; use 0 as base for carry bit; swap halves of result.
ADD	103000	Add ACS to ACD; use Carry as base for carry bit.
ADDC	103060	Add ACS to ACD; use complement of Carry as base for carry bit.
ADDCL	103160	Add ACS to ACD; use complement of Carry as base for carry bit; rotate left.
ADDCR	103260	Add ACS to ACD; use complement of Carry as base for carry bit; rotate right.
ADDCS	103360	Add ACS to ACD; use complement of Carry as base for carry bit; swap halves of result.
ADDL	103100	Add ACS to ACD; use Carry as base for carry bit; rotate left.
ADDO	103040	Add ACS to ACD; use 1 as base for carry bit.
ADDOL	103140	Add ACS to ACD; use 1 as base for carry bit; rotate left.
ADDOR	103240	Add ACS to ACD; use 1 as base for carry bit; rotate right.
ADDOS	103340	Add ACS to ACD; use 1 as base for carry bit; swap halves of result.
ADDR	103200	Add ACS to ACD; use Carry as base for carry bit; rotate right.
ADD5	103300	Add ACS to ACD; use Carry as base for carry bit; swap halves of result.
ADDZ	103020	Add ACS to ACD; use 0 as base for carry bit.
ADDZL	103120	Add ACS to ACD; use 0 as base for carry bit; rotate left.
ADDZR	103220	Add ACS to ACD; use 0 as base for carry bit; rotate right.

ADDZS	103320	Add ACS to ACD; use 0 as base for carry bit; swap halves of result.
AND	103400	And ACS with ACD; use Carry as carry bit.
ANDC	103460	And ACS with ACD; use complement of Carry as carry bit.
ANDCL	103560	And ACS with ACD; use complement of Carry as carry bit; rotate left.
ANDCR	103660	And ACS with ACD; use complement of Carry as carry bit; rotate right.
ANDCS	103760	And ACS with ACD; use complement of Carry as carry bit; swap halves of result.
ANDL	103500	And ACS with ACD; use Carry as carry bit; rotate left.
ANDO	103440	And ACS with ACD; use 1 as carry bit.
ANDOL	103540	And ACS with ACD; use 1 as carry bit; rotate left.
ANDOR	103640	And ACS with ACD; use 1 as carry bit; rotate right.
ANDOS	103740	And ACS with ACD; use 1 as carry bit; swap halves of result.
ANDR	103600	And ACS with ACD; use Carry as carry bit; rotate right.
ANDS	103700	And ACS with ACD; use Carry as carry bit; swap halves of result.
ANDZ	103420	And ACS with ACD; use 0 as carry bit.
ANDZL	103520	And ACS with ACD; use 0 as carry bit; rotate left.
ANDZR	103620	And ACS with ACD; use 0 as carry bit; rotate right.
ANDZS	103720	And ACS with ACD; use 0 as carry bit; swap halves of result.
COM	100000	Place the complement of ACS in ACD; use Carry as carry bit.

COMC	100060	Place the complement of ACS in ACD; use complement of Carry as carry bit.
COMCL	100160	Place the complement of ACS in ACD; use complement of Carry as carry bit; rotate left.
COMCR	100260	Place the complement of ACS in ACD; use complement of Carry as carry bit; rotate right.
COMCS	100360	Place the complement of ACS in ACD; use complement of Carry as carry bit; swap halves of result.
COML	100100	Place the complement of ACS in ACD; use Carry as carry bit; rotate left.
COMO	100040	Place the complement of ACS in ACD; use 1 as carry bit.
COMOL	100140	Place the complement of ACS in ACD; use 1 as carry bit; rotate left.
COMOR	100240	Place the complement of ACS in ACD; use 1 as carry bit; rotate right.
COMOS	100340	Place the complement of ACS in ACD; use 1 as carry bit; swap halves of result.
COMR	100200	Place the complement of ACS in ACD; use Carry as carry bit; rotate right.
COMS	100300	Place the complement of ACS in ACD; use Carry as carry bit; swap halves of result.
COMZ	100020	Place the complement of ACS in ACD; use 0 as carry bit.
COMZL	100120	Place the complement of ACS in ACD; use 0 as carry bit; rotate left.
COMZR	100220	Place the complement of ACS in ACD; use 0 as carry bit; rotate right.
COMZS	100320	Place the complement of ACS in ACD; use 0 as carry bit; swap halves of result.

DIA	060400	Data in, A buffer to AC.
DIAC	060600	Data in, A buffer to AC; clear device.
DIAP	060700	Data in, A buffer to AC; send special pulse to device.
DIAS	060500	Data in, A buffer to AC; start device.
DIB	061400	Data in, B buffer to AC.
DIBC	061600	Data in, B buffer to AC; clear device.
DIBP	061700	Data in, B buffer to AC; send special pulse to device.
DIBS	061500	Data in, B buffer to AC; start device.
DIC	062400	Data in, C buffer to AC.
DICC	062600	Data in, C buffer to AC; clear device.
DICP	062700	Data in, C buffer to AC; send special pulse to device.
DICS	062500	Data in, C buffer to AC; start device.
DIV	073101	If overflow, set Carry. Otherwise divide AC0-AC1 by AC2. Put quotient in AC1, remainder in AC0.
DOA	061000	Data out, AC to A buffer.
DOAC	061200	Data out, AC to A buffer; clear device.
DOAP	061300	Data out, AC to A buffer; send special pulse to device.
DOAS	061100	Data out, AC to A buffer; start device.
DOB	062000	Data out, AC to B buffer.
DOBC	062200	Data out, AC to B buffer; clear device.
DOBP	062300	Data out, AC to B buffer; send special pulse to device.
DOBS	062100	Data out, AC to B buffer; start device.
DOC	063000	Data out, AC to C buffer.

DOCC	063200	Data out, AC to C buffer; clear device.
DOCP	063300	Data out, AC to C buffer; send special pulse to device.
DOCS	063100	Data out, AC to C buffer; start device.
DSZ	014000	Decrement location E by 1 and skip if result is zero.
HALT	063077	Halt the processor (=DOC 0, CPU).
INC	101400	Place ACS + 1 in ACD; use Carry as base for carry bit.
INCC	101460	Place ACS + 1 in ACD; use complement of Carry as base for carry bit.
INCCL	101560	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; rotate left.
INCCR	101660	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; rotate right.
INCCS	101760	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; swap halves of result.
INCL	101500	Place ACS + 1 in ACD; use Carry as base for carry bit; rotate left.
INCO	101440	Place ACS + 1 in ACD; use 1 as base for carry bit.
INCOL	101540	Place ACS + 1 in ACD; use 1 as base for carry bit; rotate left.
INCOR	101640	Place ACS + 1 in ACD; use 1 as base for carry bit; rotate right.
INCOS	101740	Place ACS + 1 in ACD; use 1 as base for carry bit; swap halves of result.
INCR	101600	Place ACS + 1 in ACD; use Carry as base for carry bit; rotate right.
INCS	101700	Place ACS + 1 in ACD; use Carry as base for carry bit; swap halves of result.

INCZ	101420	Place ACS + 1 in ACD; use 0 as base for carry bit.
INCZL	101520	Place ACS + 1 in ACD; use 0 as base for carry bit; rotate left.
INCZR	101620	Place ACS + 1 in ACD; use 0 as base for carry bit; rotate right.
INCZS	101720	Place ACS + 1 in ACD; use 0 as base for carry bit; swap halves of result.
INTA	061477	Acknowledge interrupt by loading code of nearest device that is requesting an interrupt into AC bits 10-15 (=DIB -, CPU).
INTDS	060277	Disable interrupt by clearing Interrupt On (=NIOC CPU).
INTEN	060177	Enable interrupt by setting Interrupt On (=NIOS CPU).
IORST	062677	Clear all IO devices, clear Interrupt On, reset clock to line frequency (=DICC 0, CPU).
ISZ	010000	Increment location E by 1 and skip if result is zero.
JMP	000000	Jump to location E (put E in PC).
JSR	004000	Load PC + 1 in AC3 and jump to subroutine at location E (put E in PC).
LDA	020000	Load contents of location E into AC.
MOV	101000	Move ACS to ACD; use Carry as carry bit.
MOVC	101060	Move ACS to ACD; use complement of Carry as carry bit.
MOVCL	101160	Move ACS to ACD; use complement of Carry as carry bit; rotate left.
MOVCR	101260	Move ACS to ACD; use complement of Carry as carry bit; rotate right.
MOVCS	101360	Move ACS to ACD; use complement of Carry as carry bit; swap halves of result.

MOVL	101100	Move ACS to ACD; use Carry as carry bit; rotate left.
MOVO	101040	Move ACS to ACD; use 1 as carry bit.
MOVOL	101140	Move ACS to ACD; use 1 as carry bit; rotate left.
MOVOR	101240	Move ACS to ACD; use 1 as carry bit; rotate right.
MOVOS	101340	Move ACS to ACD; use 1 as carry bit; swap halves of result.
MOVR	101200	Move ACS to ACD; use Carry as carry bit; rotate right.
MOVS	101300	Move ACS to ACD; use Carry as carry bit; swap halves of result.
MOVZ	101020	Move ACS to ACD; use 0 as carry bit.
MOVZL	101120	Move ACS to ACD; use 0 as carry bit; rotate left.
MOVZR	101220	Move ACS to ACD; use 0 as carry bit; rotate right.
MOVZS	101320	Move ACS to ACD; use 0 as carry bit; swap halves of result.
MSKO	062077	Set up Interrupt Disable flags according to mask in AC (=DOB -, CPU).
MUL	073301	Multiply AC1 by AC2, add product to AC0, put result in AC0-AC1.
NEG	100400	Place negative of ACS in ACD; use Carry as base for carry bit.
NEGC	100460	Place negative of ACS in ACD; use complement of Carry as base for carry bit.
NEGCL	100560	Place negative of ACS in ACD; use complement of Carry as base for carry bit; rotate left.
NEGCR	100660	Place negative of ACS in ACD; use complement of Carry as base for carry bit; rotate right.
NEGCS	100760	Place negative of ACS in ACD; use complement of Carry as base for carry bit; swap halves of result.

NEGL	100500	Place negative of ACS in ACD; use Carry as base for carry bit; rotate left.
NEGO	100440	Place negative of ACS in ACD; use 1 as base for carry bit.
NEGOL	100540	Place negative of ACS in ACD; use 1 as base for carry bit; rotate left.
NEGOR	100640	Place negative of ACS in ACD; use 1 as base for carry bit; rotate right.
NEGOS	100740	Place negative of ACS in ACD; use 1 as base for carry bit; swap halves of result.
NEGR	100600	Place negative of ACS in ACD; use Carry as carry bit; rotate right.
NEGS	100700	Place negative of ACS in ACD; use Carry as carry bit; swap halves of result.
NEGZ	100420	Place negative of ACS in ACD; use 0 as base for carry bit.
NEGZL	100520	Place negative of ACS in ACD; use 0 as base for carry bit; rotate left.
NEGZR	100620	Place negative of ACS in ACD; use 0 as base for carry bit; rotate right.
NEGZS	100720	Place negative of ACS in ACD; use 0 as base for carry bit; swap halves of result.
NIO	060000	No operation.
NIOC	060200	Clear device.
NIOP	060800	Send special pulse to device.
NIOS	060100	Start device.
READS	060477	Read console data switches into AC (=DIA -, CPU).
SBN	000007	Skip if both carry and result are nonzero (skip function in an arithmetic or logical instruction).

SEZ	000006	Skip if either carry or result is zero (skip function in an arithmetic or logical instruction).
SKP	000001	Skip (skip function in an arithmetic or logical instruction).
SKPBN	063400	Skip if Busy is 1.
SKPBZ	063500	Skip if Busy is 0.
SKPDN	063600	Skip if Done is 1.
SKPDZ	063700	Skip if Done is 0.
SNC	000003	Skip if carry bit is 1 (skip function in an arithmetic or logical instruction).
SNR	000005	Skip if result is nonzero (skip function in an arithmetic or logical instruction).
STA	040000	Store AC in location E.
SUB	102400	Subtract ACS from ACD; use Carry as base for carry bit.
SUBC	102460	Subtract ACS from ACD; use complement of Carry as base for carry bit.
SUBCL	102560	Subtract ACS from ACD; use complement of Carry as base for carry bit; rotate left.
SUBCR	102660	Subtract ACS from ADC; use complement of Carry as base for carry bit; rotate right.
SUBCS	102760	Subtract ACS from ACD; use complement of Carry as base for carry bit; swap halves of result.
SUBL	102500	Subtract ACS from ACD; use Carry as base for carry bit; rotate left.
SUBO	102440	Subtract ACS from ACD; use 1 as base for carry bit.
SUBOL	102540	Subtract ACS from ACD; use 1 as base for carry bit; rotate left.
SUBOR	102640	Subtract ACS from ACD; use 1 as base for carry bit; rotate right.

SUBOS	102740	Subtract ACS from ACD; use 1 as base for carry bit; swap halves of result.
SUBR	102600	Subtract ACS from ACD; use Carry as base for carry bit; rotate right.
SUBS	102700	Subtract ACS from ACD; use Carry as base for carry bit; swap halves of result.
SUBZ	102420	Subtract ACS from ACS; use 0 as base for carry bit.
SUBZL	102520	Subtract ACS from ACD; use 0 as base for carry bit; rotate left.
SUBZR	102620	Subtract ACS from ACD; use 0 as base for carry bit; rotate right.
SUBZS	102720	Subtract ACS from ACD; use 0 as base for carry bit; swap halves of result.
SZC	000002	Skip if carry is 0 (skip function in an arithmetic or logical instruction).
SZR	000004	Skip if result is zero (skip function in an arithmetic or logical instruction).
@	002000	When this character appears in an instruction, the assembler places a 1 in bit 5 to produce indirect addressing.
@	100000	When this character appears with a 15-bit address, the assembler places a 1 in bit 0, making the address indirect.
#	000010	Appending this character to the mnemonic for an arithmetic or logical instruction places a 1 in bit 12 to prevent the processor from loading the 17-bit result in Carry and ACD. Thus the result of an instruction can be tested for a skip without affecting Carry or the accumulators.

APPENDIX B

BASIC A/D AND D/A CONVERSION

In A/D and D/A conversion systems, the information to be converted is mainly in two forms. One form is an analog signal on a single line whose magnitude represents some physical quantity. The second form is discrete bits of information, either serial on a single line or parallel on many lines which describe a weighted digital word. The word, of course, is the description of the quantity.

An example here will serve to introduce the nomenclature associated with these techniques. Figure B-1 shows a three bit binary word and its value in decimal form.

Digital Lines	Least Value Word	Increasing Value							Greatest Value Word
$\text{Value}=2^2=4$ <u>M. S. B.</u>	0	0	0	0	1	1	1	1	
$\text{Value}=2^1=2$ <u>N. M. S. B.</u>	0	0	1	1	0	0	1	1	
$\text{Value}=2^0=1$ <u>L. S. B.</u>	0	1	0	1	0	1	0	1	
Analog Voltage	0V	2V	4V	6V	8V	10V	12V	14V	

Figure B-1. Example Digital Coding

The example uses volts as the quantity represented, and two volts has been assigned as the value for the least significant bit (LSB). Since the smallest quantum that can be resolved is 2 volts, it is possible to represent up to 14v in 2 volt increments. The resolution of the digital word and its analog equivalent is one part in seven, or 2 volts per step. A quantum is defined as the smallest quantity that can be represented by a digital word and is equal to the LSB.

Now if we can add a sign bit before the most significant bit (MSB) and use the same total space of voltage (i. e., 14 volts), we can represent analog voltages from +7 volts to -7 volts with a 1 volt increment, as shown in Figure B-2.

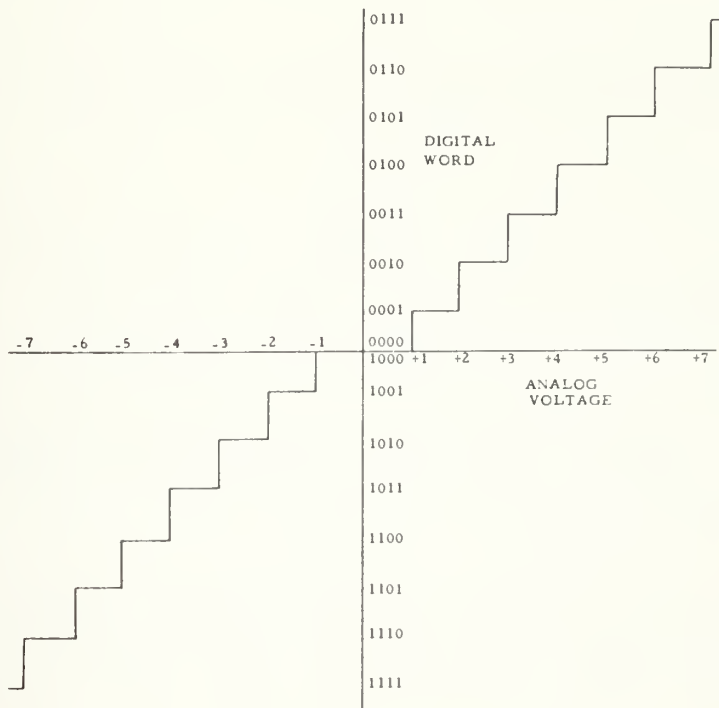


Figure B-2 Digital Word vs. Equivalent Analog Voltage

The reason for handling data in a digital fashion in some systems is the greater speed and accuracy attainable and reduction of the effect of noise on system accuracy. Also, from a reliability standpoint, digital equipment can be made redundant; for example, majority voting techniques can be used; this is much more difficult in analog equipment. However, the analog form is more than likely needed somewhere in the system because of the difficulty in manufacturing transducers which respond to digital formats.

B.1. Basic D/A Conversion

Figure B-3 shows a three bit, signed converter. The basic elements are the storage register, the reference supply, and the D/A decoder. The input word is stored in the register. Each flipflop drives an analog switch which connects the reference supply voltage or ground to the proper terminal of the resistor network. In the case of a "1" bit, the resistor network divides the reference voltage down so that a voltage is added to the D/A output whose magnitude is proportional to the equivalent weight of this particular bit. The sign bit controls the polarity of reference voltage to be applied to the resistor network. In cases where accuracy required is extremely low, the storage resistor flipflop could be used to drive the resistor network directly, but, of course, this is not normally the case. The accuracy of this system would be dependent upon a number of error sources.

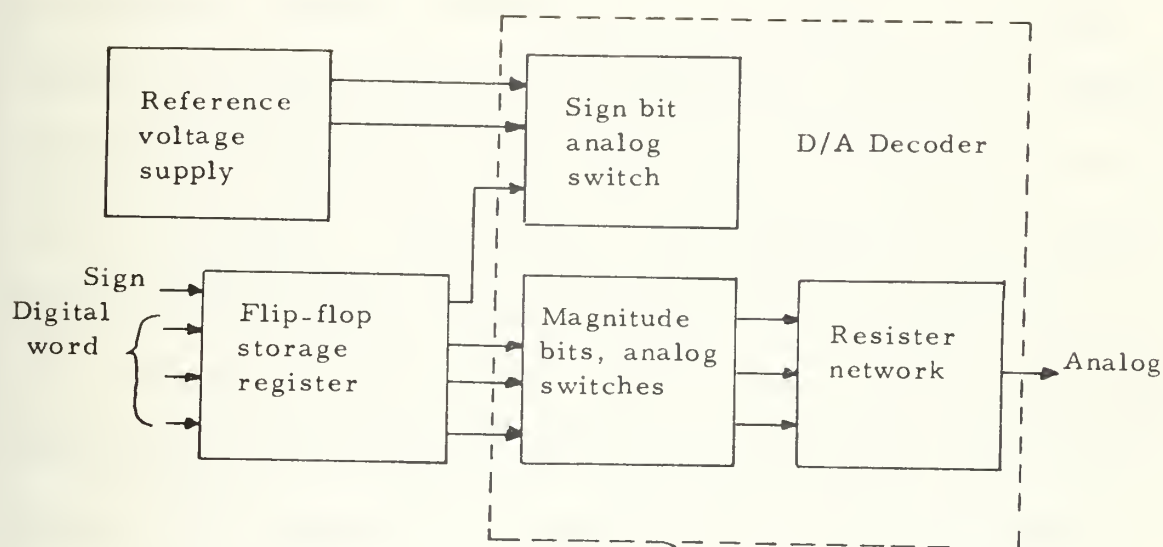


Figure B-3 Basic D/A Conversion

A more formidable problem exists in A/D conversion than does in D/A. The reasons are: (1) The A/D converter represents the front end of the digital processing, and thus the generation of errors are much more critical at this stage; (2) The A/D process normally takes longer than the D/A, so this stage must meet the sampling rate requirements; (3) High sampling rates and increased accuracy give rise to more complex equipment. To illustrate these statements we will consider three examples.

B.2. Successive Approximations A/D Conversion System

Figure B-4 shows a typical A/D conversion system of a type referred to as serial or successive approximations. The process is as follows: a multiplexer channel is selected by the programmer. The analog voltage of the channel appears at the input of the comparator. Simultaneously, the digital register is cleared and all "0's appear at the input of the D/A decoder. The comparator then compares the zero voltage from the D/A decoder with the multiplexer output and determines the sign. The proper sign is then set into the register by the programmer, which then sets a "1" into the MSB of the D/A decoder. The decoder output is then compared to the analog voltage to determine which is larger. If the D/A converter is largest, the one is removed from the MSB of the decoder. If the analog voltage is larger, the MSB is left alone. The process continues until a "1" has been tried in each succeeding less significant bit. The digital equivalent is then read out of the register.

B.3. Counter Ramp A/D Conversion

We would like to reduce the complexity of the previous example and still maintain an adequate conversion time. One way is illustrated in Figure B-5. The process begins by setting the counter to all "0"s. The clock pulses are then counted up in the register which drives the D/A decoder to follow the count with an analog voltage. As the count

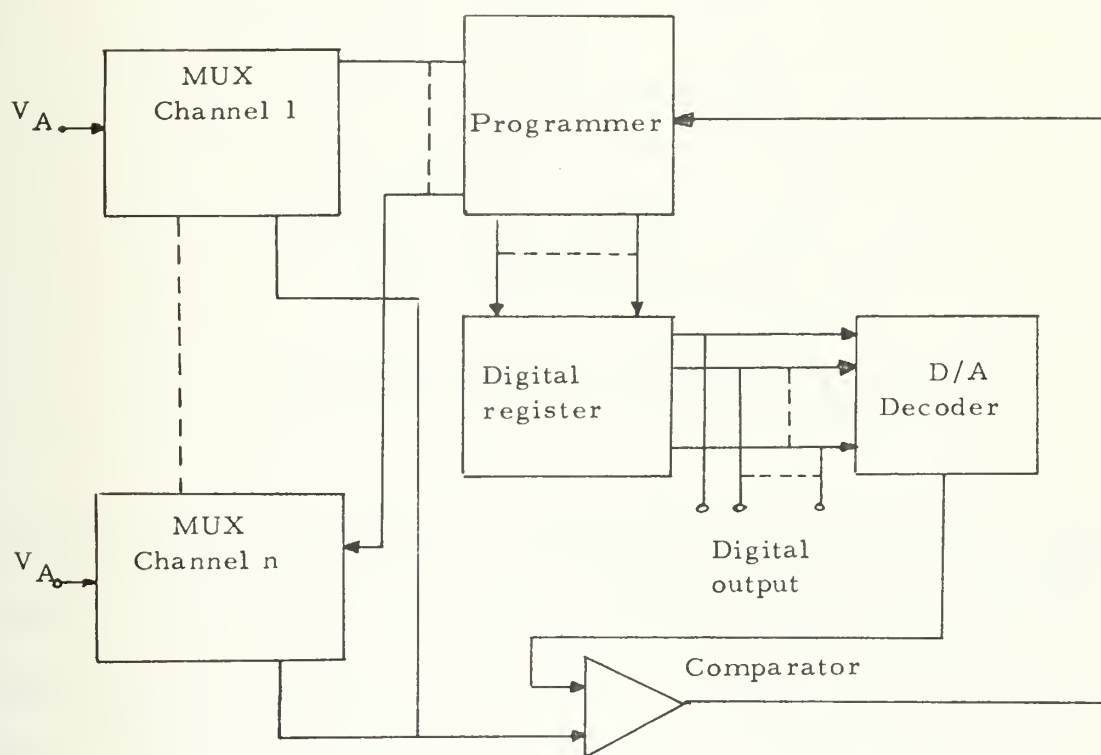


Figure B-4 Successive Approximation A/D Converter

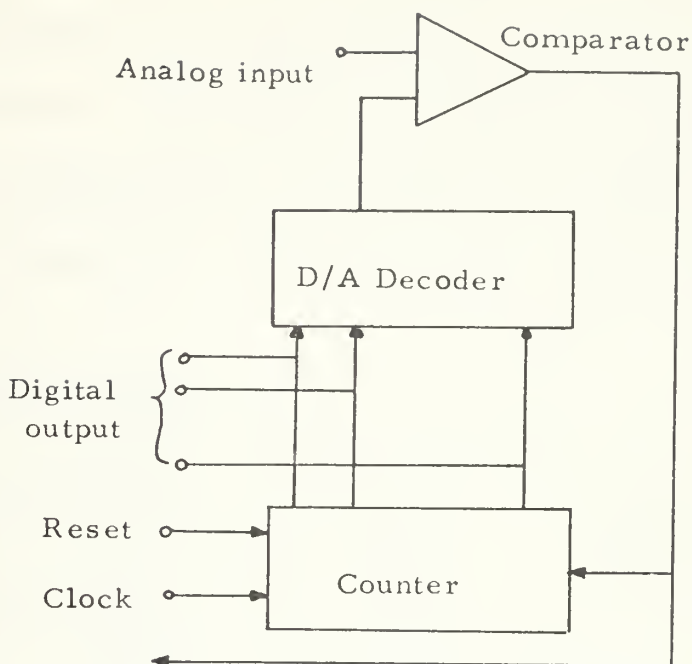


Figure B-5 Counter Ramp A/D Converter

increases, the voltage from the decoder increases linearly with time until the ramp voltage is equal to or slightly greater than the analog input signal. At this point the comparator signals the counter to stop, and the digital word is read from the counter. This circuit appears to have reduced the complexity of the previous example, but it has introduced a rather high speed clock, which certainly won't reduce the cost efficiency.

B.4. Parallel A/D Conversion

This is the fastest type of A/D conversion, but it sacrifices simplicity for increased speed. The circuit is shown in Figure B-6. The

circuit shown is for a 3 bit conversion, and as shown must have enough threshold detectors to cover all possible digital numbers (i. e., 3 bits = 7 possible numbers). The analog input appears at the input of all threshold detectors simultaneously. The detectors drive the encoding logic which looks for the highest reference exceeded, and encodes accordingly.

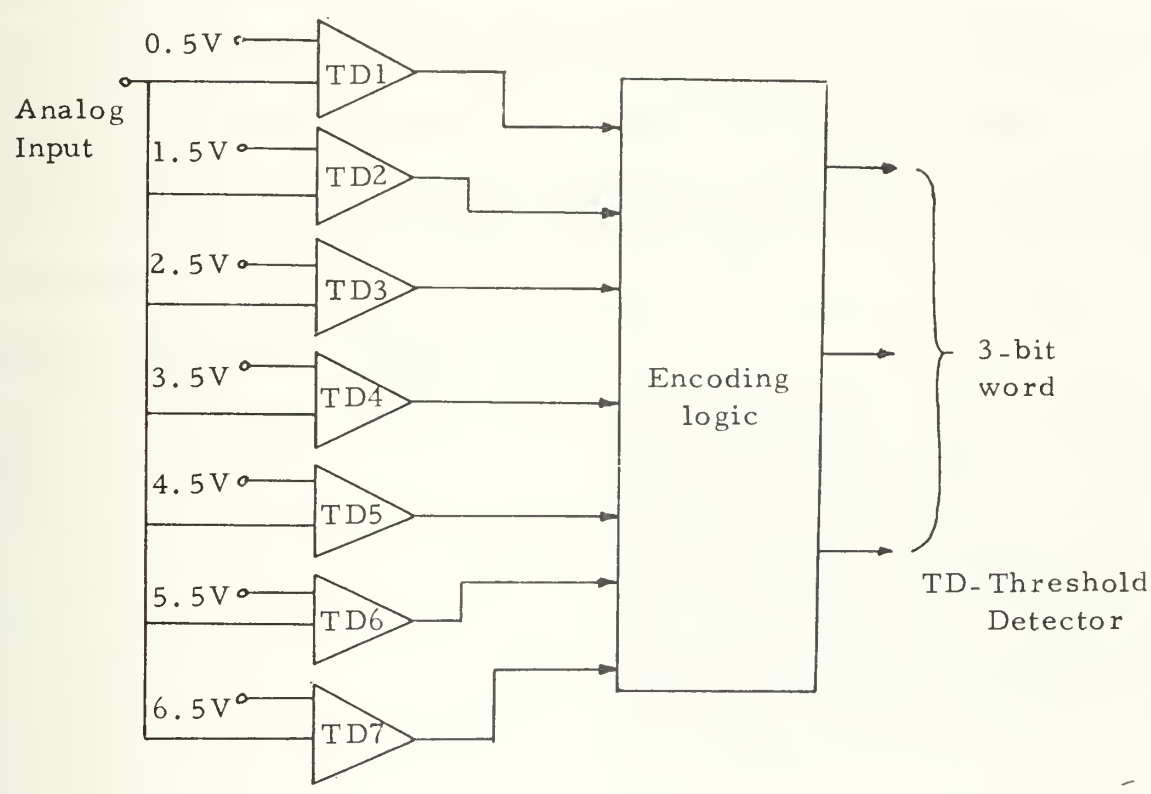


Figure B-6 Parallel A/D Conversion

APPENDIX C

BUS SIGNALS

The binary signals on the bus have two states, low and high, which correspond respectively to nominal voltage levels of 0 and +2.7 volts. Any level between ground and .4 volt is interpreted as low and level more positive than 2.2 volts is interpreted as high. The level listed for a signal in the following table is the voltage level on the line when the signal represents a 1 or produces the indicated function. A low signal is indicated in the prints by a bar over its name.

<u>Signal</u>	<u>Direction</u>	<u>Level</u>	
DS0 to DS5	To device	Low	Device Selection. The processor places the device code (bits 10-15 of the instruction word) on these lines during the execution of an in-out instruction. The lines select one of 62 devices (codes 01-76) that may be connected to the bus. Only the selected device responds to control signals generated during the instruction.
DATA0 to DATA15	Bydirectional	Low	Data. All data and addresses are transferred between the processor and the devices attached to the bus via these sixteen lines. For programmed output the processor places the AC specified by the instruction on the data lines and then generates DATOA, DATOB or DATOC to load the data from the lines into the corresponding buffer in the device selected by DS0-5, or generates MSKO to set up the Interrupt Disable flags in all of the devices according to the mask on the data lines. For data channel output the processor places the memory buffer

on the data lines and generates DCHO to load the contents of the lines into the data buffer in the device that is being serviced.

For programmed input the processor generates DATIA, DATIB or DATIC to place information from the corresponding buffer in the device selected by DS0-5 on the data lines, or generates INTA to place the code of the nearest device that is requesting an interrupt on lines 10-15. The processor then loads the data from the lines into the AC selected by the instruction. To get an address for data channel access the processor generates DCHA to place a memory address from the nearest device that is requesting access on lines 1-15 and then loads the address into the memory address register. For data channel input the processor generates DCHI to place the data buffer of the device being serviced on the data lines and then loads the contents of the lines into the memory buffer.

DATOA	To device	High	Data Out A. Generated by the processor after AC has been placed on the data lines in a DOA to load the data into the A buffer in the device selected by DS0-5.
DATIA	To device	High	Data In A. Generated by the processor during a DIA to place the A buffer in the device selected by DS0-5 on the data lines.
DATOB	To device	High	Data Out B. Equivalent to DATOA but loads the B buffer.
DATIB	To device	High	Data In B. Equivalent to DATIA but places the B buffer on the data lines.
DATOC	To device	High	Data Out C. Equivalent to DATOA but loads the C buffer.
DATIC	To device	High	Data In C. Equivalent to DATIA but places the C buffer on the data lines.

STRT	To device	High	Start. Generated by the processor in any nonskip IO instruction with an S control function (bits 8-9=01) to clear Done, set Busy, and clear the INT REQ flipflop in the device selected by DS0-5.
CLR	To device	High	Clear. Generated by the processor in any nonskip IO instruction with a C control function (bits 8-9=10) to clear Busy, Done and the INT REQ flipflop in the device selected by DS0-5.
IOPLS	To device	High	IO Pulse. Generated by the processor in any nonskip IO instruction with a P control function (bits 8-9=11) to perform some special function in the device selected by DS0-5 (this signal is for custom applications).
SELB	To processor	Low	Selected Busy. Generated by the device selected by DS0-5 if its Busy flag is set.
SELD	To processor	Low	Selected Done. Generated by the device selected by DS0-5 if its Done flag is set.
RQENB	To device	Low	Request Enable. Generated at the beginning of every memory cycle to allow all devices on the bus to request program interrupts or data channel access. In any device RQENB sets the INT REQ flipflop if Done is set and Interrupt Disable is clear. Otherwise it clears INT REQ. In any device connected to the data channel RQENB sets the DCH REQ flipflop if the DCH SYNC flipflop is set. Otherwise it clears DCH REQ.
INTR	To processor	Low	Interrupt Request. Generated by any device when its INT REQ flipflop is set. This informs the processor that the device is waiting for an interrupt to start.

INTP	To device	Low	Interrupt Priority. Generated by the processor for transmission serially to the devices on the bus. If the INT REQ flipflop in a device is clear when the device receives INTP, the signal is transmitted to the next device.
INTA	To device	High	Interrupt Acknowledge. Generated by the processor during the INTA instruction. If a device receives INTA while it is also receiving INTP and its INT REQ flipflop is set, it places its device code on data lines 10-15.
MSKO	To device	Low	Mask Out. Generated by the processor during the MSKO instruction after AC has been placed on the data lines to set up the Interrupt Disable flags in all devices according to the mask on the lines.
DCHR	To processor	Low	Data Channel Request. Generated by any device when its DCH REQ flipflop is set. This informs the processor that the device is waiting for data channel access.
DCHP	To device	Low	Data Channel Priority. Generated by the processor and transmitted serially to the devices on the bus. If the DCH REQ flipflop in a device is clear when the device receives DCHP, the signal is transmitted to the next device.
DCHA	To device	Low	Data Channel Acknowledge. Generated by the processor at the beginning of a data channel cycle. If a device receives DCHA while it is also receiving DCHP and its DCH REQ flipflop is set, it places the memory address to be used for data channel access on data lines 1-15 and sets its DCH SEL flipflop.
DCHM0	To processor	Low	Data Channel Mode. Generated by a device when its DCH

DCHM1

SEL flipflop is set to inform the processor of the type of data channel cycle desired as follows:

DCHM	DCHM1	
0(H)	0(H)	Data out
0(H)	1(L)	Increment memory
1(L)	0(H)	Data in
1(L)	1(L)	Add to memory

In addition to performing the necessary functions internally, the processor generates DCHI and/or DCHO for the required in-out transfers.

DCHI To device High

Data Channel In. Generated by the processor for data channel input (DCHM0=1) to place the data register of the device selected by DCHA on the data lines.

DCHO To device High

Data Channel Out. Generated by the processor for data channel output (DCHM0-1≠10) after the word from memory or the arithmetic result has been placed on the data lines to load the contents of the lines into the data register of the device selected by DCHA.

OVFLO To device High

Overflow. Generated by the processor during a data channel cycle that increments memory or adds to memory (DCHM1=1) when the result exceeds $2^{16} - 1$.

IORST To device High

IO Reset. Generated by the processor in the IORST instruction or when the console reset switch is pressed to clear the control flipflops in all interfaces connected to the bus. This signal is also generated during power turnon.

APPENDIX D

DEVICE CONNECTION AND SPECIFICATION

D.1 Device Connection

1. A/D Converter

<u>Pin</u>	<u>Connection</u>	<u>Remarks</u>
1U	+15 volts	
2U	12U	Required for <u>+10v</u> operation
3U	Channel 0 in	
4U	Channel 1 in	
5U	Channel 2 in	
6U	Channel 3 in	
7U	Channel 4 in	
8U	Channel 5 in	
9U	Channel 6 in	
10U	Channel 7 in	
11U	17U	To complete signal circuit
12U	2U	Required for <u>+10v</u> operation
13U	14U	Required for <u>+10v</u> operation
14U	13U	
15U	14L	Required for <u>+10v</u> operation
16U	NC	
17U	11U	To complete signal circuit

18U	NC	
19U	NC	
20U	NC	
21U	NC	
22U	NC	
23U	27U	Starts A/D conversion when MUX has settled
24U	44	$\overline{\text{DATOC}}$ (start conversion)
25U	+5v	Enables strobe A to operate
26U	+5v through 3.9K resistor	Allows A/D TRIG (-) to operate
27U	23U	Starts A/D conversion when MUX has settled
28U	DIGITAL GROUND	To specify 12 bit word length
29U	NC	
30U	130A	BIT 3
31U	90	BIT 5
32U	130	BIT 7
33U	90A	BIT 9
34U	78A	BIT 11
35U	DIGITAL GROUND	
36U	+5v	
1L	-15v	
2L	NC	
3L	Channel 0 return	

4L	Channel 1 return	
5L	Channel 2 return	
6L	Channel 3 return	
7L	Channel 4 return	
8L	Channel 5 return	
9L	Channel 6 return	
10L	Channel 7 return	
11L	12L	To complete signal return circuit
12L	11L	
13L	NC	
14L	13U	required for <u>+10v</u> operation
15L	NC	
16L	NC	
17L	NC	
18L	NC	
19L	NC	
20L	84	
21L	87	Channel address bits
22L	86A	
23L	NC	
24L	DIGITAL GROUND	
25L	NC	
26L	NC	
27L	N74123-PIN 9	End of conversion

28L	96	$\overline{\text{MSB}}$
29L	78	BIT 2
30L	73	BIT 4
31L	105	BIT 6
32L	74A	BIT 8
33L	106A	BIT 10
34L	111	LSB
35L	DIGITAL GROUND	
36L	+5v	

2. D/A Converter

<u>Pin</u>	<u>Connection</u>	<u>Remarks</u>
1U	+15v	
2U	2L & Digital Ground	Analog return
3U	7U	For <u>+10v</u> operation
4U	NC	
5U	NC	
6U	3L	For <u>+10v</u> operation
7U	3U & 10U	For <u>+10v</u> operation
8U	11U	For <u>+10v</u> operation
9U	XO wiper	X channel offset adjust
10U	7U	For <u>+10v</u> operation
11U	8U	For <u>+10v</u> operation

12U	NC	
13U	NC	
14U	NC	
15U	NC	
16U	NC	
17U	NC	
18U	NC	
19U	NC	
20U	112	BIT 4
21U	118	BIT 3
22U	112A	BIT 2
23U	121	BIT 1
24U	NC	
25U	NC	
26U	NC	
27U	NC	
28U	7400-PIN3	DATOB
29U	7400-PIN11	<u>DS23</u>
30U	NC	
31U	NC	
32U	NC	
33U	NC	
34U	NC	

35U	DIGITAL RETURN	
36U	+5v	
1L	-15v	
2L	2U & Digital Ground	Analog Return
3L	6U	For <u>+</u> 10v operation
4L	NC	
5L	YO wiper	Y channel offset adjust
6L	9L & 2U	Ties X, Y, and analog returns together
7L	YG wiper	Y channel gain adjust
8L	XG wiper	X channel gain adjust
9L	6L & 2U	Ties X, Y, and analog return together
10L	NC	
11L	NC	
12L	NC	
13L	NC	
14L	NC	
15L	NC	
16L	NC	
17L	27L	Ties X-Y select Bit IN to LSB
18L	NC	
19L	NC	
20L	94A	BIT 5

21L	92A	BIT 6
22L	94	BIT 7
23L	91	BIT 8
24L	88A	BIT 9
25L	84	BIT 10
26L	87	BIT 11
27L	86A & 17L	LSB
28L	NC	
29L	7400-PIN 6	DATOA
30L	DIGITAL GROUND	DISABLES MASTER CLEAR
31L	NC	
32L	NC	
33L	NC	
34L	NC	
35L	DIGITAL GROUND	
36L	+5v	

3. N 7400

<u>Pin</u>	<u>Connection</u>	<u>Remarks</u>
1	14A	<u>DATOB</u> IN
2	14A	
3	D/A-PIN28U	DATOA OUT - STROBE FOR INPUT DATA
4	18	<u>DATOA</u> IN
5	18	

6	D/A-PIN29L	DATOB OUT - STROBE FOR X-Y SELECT
7	DIGITAL GROUND	
8	NC	$\overline{DS21}$ - Not Needed
9	U15-PIN10	$\overline{DS4}$ IN
10	50	DS21+DS23 IN
11	D/A-PIN29U	$\overline{DS23}$ OUT - D/A DEVICE SELECT
12	50	DS21+DS23 IN
13	U15-PIN4	DS4 IN
14	+5v	

4. N 74123

<u>Pin</u>	<u>Connection</u>	<u>Remarks</u>
1	NC	
2	NC	
3	NC	
4	NC	
5	NC	
6	Through capacitor to Pin 7	C=.001ufd
7	Through capacitor to Pin 6 through resistor to +5v	R=10K
8	DIGITAL GROUND	
9	A/D-PIN27L	EOC - IN
10	+5v	
11	+5v	

12	53A	SHORTENED EOC PULSE - TO SET DONE FLIPFLOP
13	NC	
14	NC	
15	NC	
16	+5v	

D.2 DT 1620 Functional Pin Description

<u>Pin</u>	<u>Remarks</u>
1U	+15 volts @ 80 mA maximum, 0.1% regulation
1L	-15 volts @ 80 mA maximum, 0.1% regulation
2U, 2L	Analog return (internally tied to Digital return at one point in the comparator)
3U	Channel 0 return on Differential units only
4U	Ch. 1 signal input. Use 4L for Ch. 1 return on Differential units only
5U	Ch. 2 signal input. Use 5L for Ch. 2 return on Differential units only
6U	Ch. 3 signal input. Use 6L for Ch. 3 return on Differential units only
7U	Ch. 4 signal input. Use 7L for Ch. 4 return on Differential units only
8U	Ch. 5 signal input. Use 8L for Ch. 6 return on Differential units only
9U	Ch. 6 signal input. Use 9L for Ch. 6 return on Differential units only
10U	Ch. 7 signal input. Use 10L for Ch. 7 return on Differential units only

- 3L Ch. 8 signal input for single ended units only.
3L is return line for Ch. 0 on Differential units only
- 4L Ch. 9 signal input for single ended units only.
4L is return line for Ch. 1 on Differential units only
- 5L Ch. 10 signal input for single ended units only.
5L is return line for Ch. 2 on Differential units only
- 6L Ch. 11 signal input for single ended units only.
6L is return line for Ch. 3 on Differential units only
- 7L Ch. 12 signal input for single ended units only.
7L is return line for Ch. 4 on Differential units only
- 8L Ch. 13 signal input for single ended units only.
8L is return line for Ch. 5 on Differential units only
- 9L Ch. 14 signal input for single ended units only.
9L is return line for Ch. 6 on Differential units only
- 10L Ch. 15 signal input for single ended units only.
10L is return line for Ch. 7 on Differential units only
- 11U Multiplexer output. HI side on Differential units (LO side is 11L) when using switch gain amplifier or other gain element external to the unit 11U must not "see" an impedance 100 Megohm to avoid loading errors.
- 11L No connection on this pin with single ended units.
Represents LO side of signal on Differential units.
Must not be loaded with an impedance of 100 Megohms.
- 12U Determines gain of Differential Amplifier. If tied to analog return (2U, 2L) Differential Amplifier has gain of -1 for 0-10v, +10 volt operation. When tied to 13L, Differential Amplifier has gain of -2 for 0-5v, +5 volt operation.
- 12L LO side of signal input to Differential Amplifier normally tied to signal commons on single ended units; Tied to 11L (LO side MUX out) on Differential units.
- 13U Output of Sample/Hold. (S&H inverts).

- 13L Output of Differential Amplifier. (Differential Amplifier inverts).
- 14U Range 1 input on A/D (20v range) when tied to range 2 (14L) gives 10 volts range.
- 14L Range 1 input on A/D.
- 15U +10v reference. When tied to 14U or 14L, offsets A/D $\frac{1}{2}$ scale for Bipolar ranges. Also used for remote range adjust Pot can not deliver more than 4 mA.
- 15L Optional. For "CS" units this Pin has A/D clock signal.
- 16U For remote Pot used for offset adjust when internal Pot not accessible.
- 16L Reference adjust. For remote Pot when internal range adjust is not used. Adjusts reference and full scale of A/D.
- 17U Used only with option "SG". Brings out the HI input side of the Differential Amplifier separated from MUX.
- HI HI output. For use when inserting external amplifier between MUX and Differential Amplifier.
- 17L MUX Enable. Internally pulled HIGH (with 3.3K). Used only with expander to disable (LO) Bank 1. Normally left open (HI).
- 18U Connected to A/D serial output on option "CS" only. Data is inverted (LO=1).
- 18L Used with option "SH" to bring sample/hold control line out separate from internal timing control. HIGH = Sample, LO = Hold. 1 TTL Load.
- 19U Address Bit Output A_8 on single ended units only. Not used on Differential units. Internally pulled up with 6.8K 10 TTL loads.
- 20U Address Bit output A_4 (SE and DI). Pulled up internally with 6.8K 10 TTL loads.

- 21U Address Bit output A_2 (SE and DI). 6.8K Internal pull up, 10 TTL loads.
- 22U Address Bit output A_1 . 6.8K internal pull up 10 TTL loads.
- 19L Address Bit A_8 input. No internal pull up 1 TTL load. Not used on Differential units. Must be stable for 100ns after Strobe.
- 20L Address Bit A_4 input. No internal pull up 1 TTL load. Must be stable for 100ns after Strobe.
- 21L Address Bit A_2 input. No internal pull up 1 TTL load. Must be stable for 100ns after Strobe.
- 22L Address Bit A_1 input. No internal pull up 1 TTL load. Must be stable for 100ns after Strobe.
- 23U MUX S&H time-out. Goes HI at Strobe, goes LO when MUX, S&H have settled. (Duration: 4us DT1610, 7us DT1620, 15us DT1640). Drives 10 TTL loads. Can be used to automatically trigger an A/D conversion after MUX, S&H have settled by tieing to Pin 27U, A/D trigger (-).
- 23L For use with external Pot when MUX S&H timeout is to be adjusted.
- 24U Strobe A. HI to LO transition causes MUX channel select (sequential or random and causes S&H to sample). Initiates time-out function. Internally pulled up with 3.3K.
- 25U Strobe B. Does identical function as Strobe A. Can be used to inhibit Strobe A by grounding (LO). Must be HI to Enable Strobe A.
- 24L Load Enable. When LO, at Strobe the MUX will load the channel address presented to Pin 19L thru 22L. (Clear Enable must be HI). When Load Enable is HI, the MUX will advance one channel from where it was at Strobe. No internal pull up.
- 25L Clear Enable. When LO, will cause MUX to go to Ch.0 at Strobe independent of Load Enable. When HI does nothing. Internally pulled up with 6.8K.

- 26U A/D Trig (+). A positive edge (LO to HI transition) will start a conversion provided A/D Trig (-), Pin 27U is LO. No internal pull up. May be used to inhibit A/D Trig (-) by grounding 26U.
- 27U A/D Trig (-). A HI to LO transition will cause a conversion provided A/D Trig (+) is HI. Both A/D Trig (+) and A/D Trig (-) must not be activated during a conversion.
- 26L Clock Adjust for remote Pot. to adjust A/D clock frequency, if required.
- 27L End of Conversion. Goes HI when A/D is triggered and remains HI until conversion is complete and data is valid in the output storage register at which time EOC goes LO and remains LO until next A/D Trigger. 10 TTL loads.
- 28U Word Length. Must be grounded (Digital Ground) for 12 bits. If left open short cycles converter to one bit! Tie to bit N+1 for N bit word, i. e., tie to bit 11 Pin 34U for 10 bit converted word.
- 28L $\overline{\text{MSB}}$. Complement of MSB. Use as MSB for proper 2's complement coding. 10 TTL loads.
- 29U MSB. Use as MSB for offset and natural binary coding. 10 TTL loads.
- 29L, 30U, 30L, 31U, 31L, 32U 32L, 33U, 33L, 34U, 34L Bit 2 (next MSB), Bit 3, Bit 4, Bit 5, Bit 6, But 7, Bit 8, Bit 9, Bit 10, Bit 11, Bit 12(LSB), respectively. All these bit outputs can drive 10 TTL loads.
- 35U, 35L Digital Return.

APPENDIX E

ASSEMBLER ERROR FLAGS

Error Flag	General Class of Problem	Examples/Comments	
A	Address error	LDA 0, 400 ISZ . +317	
B	Bad character	LA\$L: LDA 1, 23 ; \$ Not permitted	
C	Colon error	A+2: ; no expression permitted before a colon	
D	Radix error	.RDX 12 ; Radix 12 not permitted	
E	Equal error	REG= 3+B ; Where B is undefined	
F	Format error	ADD 2 ; Need at least 2 operands	
G	Declaration error	Error in declaration of an internal or external symbol	
I	Input error	Parity checked on input and some character was in error	
K	Conditional error	Conditional assembly error - Expression used in .IFE or .IFN pseudo-ops is not evaluable in pass 1, or the .IFE or .IFN pseudo-op is nested within a previous conditional assembly statement.	
L	.LOC error	.LOC -1 ; Bit 0 set	
M	Multiply defined symbol	A: 3 ; Symbol may appear only once in label field A: 5	
N	Number error	C77: 7A ; No letters permitted in a number	
O	Field overflow	LDA 4, LOC ; No register 4	
P	Phase error	Value of a symbol in pass 1 differs from that of pass 2	

Q	Questionable line	.+.END	
R	Expression error	Expression error - expression does not evaluate to be absolute, relocatable, or byte pointer type relocatable, or expression mixes page zero and normal relocatable symbols incorrectly	
S	Symbol table overflow	Memory capacity for a given machine has reached	
T	Error in table pseudo-op	14.+.XPNG	; No expression before a table pseudo-op
U	Undefined symbol	A symbol in operand field was never defined	
X	Text error	LET. "CB"	; Only one character in " Atom No expression permissible before .TXT
Z	Symbol error	Expression contains illegal symbol, (e.g., an external, an op code, double precision number, or floating point number).	

APPENDIX F

MANUFACTURER SUPPLIED SUBROUTINES

The following subroutines are reprinted with permission of Data General Corporation

; NAME: DBIN.SR PART NUMBER: 090-000029

; DESCRIPTION: DECIMAL TO BINARY S.P.

; REVISION HISTORY:

REV.	DATE
00	6/6/69
01	11/16/73

; COPYRIGHT (C) DATA GENERAL CORPORATION, 1969, 1973
; ALL RIGHTS RESERVED.

; LICENSED MATERIAL - PROPERTY OF DATA GENERAL
; CORPORATION

; CONVERT AN ASCII CHARACTER STRING TO A SINGLE
; PRECISION BINARY NUMBER
; CONVERTS AN ASCII DECIMAL CHARACTER STRING TO A
; TWO'S COMPLEMENT, FIXED POINT, BINARY NUMBER

; INPUT: CALLS A GET CHARACTER ROUTINE WHOSE
; ADDRESS MUST BE STORED IN LOCATION 40
; OF PAGE 0
; CHARACTERS MUST BE RETURNED RIGHT
; ADJUSTED IN AC0 WITH BIT 8=0
; + IS OPTIONAL FOR POSITIVE NUMBERS
; - MUST BE GIVEN FOR NEGATIVE NUMBERS
; INPUT OF FORM:
; SDD...D(BREAK)
; S IS THE SIGN, D A DECIMAL DIGIT
; THE BREAK CHARACTER IS ANY CHARACTER
; OTHER THAN A DIGIT.

; OUTPUT: AC0 CONTAINS THE BREAK CHARACTER
; AC1 CONTAINS THE BINARY INTEGER
;


```

; CALLING SEQUENCE:
;   JSR      .DBIN
;   RETURN
;
; IF AN INDICATION IS DESIRED TO SIGNAL CHARACTERS ARE
; REQUESTED, CALLING SEQUENCE IS:
;   JSR      .DBNI
;   RETURN
;
; AN ASCII "S" FOLLOWED BY A NULL WORD
; WILL BE TRANSMITTED VIA AC0 TO A PUT CHARACTER
; ROUTINE WHOSE ADDRESS MUST BE IN LOCATION 41 OF PAGE 0
;
; CAUTION:      THE ABSOLUTE VALUE OF THE RESULT IS
;                N MOD 2**15.
;                FOR EXAMPLE, +96741 CONVERTS TO +31205
;                -2**15 CONVERTS TO 0
;
; DESTROYED: AC0, AC1
; UNCHANGED: AC2, AC3

.DBNI:  STA 3, .EC03      ; SAVE RETURN
        STA 2, .EC02      ; SAVE AC2
        LDA 0, .EC24      ; GET "S"
        JSR @.EC41        ; SEND "S"
        SUB 0, 0
        JSR @.EC41        ; SEND NULL
        JMP .+3

.DBIN:  STA 3, .EC03      ; SAVE AC3
        STA 2, .EC02      ; SAVE AC2
        SUB 0, 0
        STA 0, .EC10      ; CLEAR SIGN WORD
        STA 0, .EC11      ; CLEAR SUM WORD
        JSR @.EC40        ; GET A CHARACTER
        LDA 1, .EC20      ; TEST FOR "+"
        SUB 0, 1, SNR
        JMP .EC97         ; YES
        LDA 1, .EC21      ; NO, TEST FOR "-"
        SUB 0, 1, SZR
        JMP .EC96         ; NO EXPLICIT SIGN
        ISZ .EC10         ; SET FLAG WORD FOR NEGATIVE
                           ; NUMBER
.EC97:  JSR @.EC40        ; GET ANOTHER CHARACTER
.EC96:  LDA 1, .EC22      ; ASCII "0"
        LDA 2, .EC23      ; ASCII "9"
        ADCZ# 2, 0, SNC   ; SKIP IF      9
        ADCZ# 0, 1, SZC   ; SKIP IF     = 0

```



```

        JMP .EC95          ; NOT A DIGIT, THEREFORE A BREAK
                           ; CHARACTER
        SUB 1,0            ; REDUCE DIGIT TO 0-9 BINARY
                           ; RANGE
        LDA 1,.EC11        ; SUM WORD
        JSR .EC50          ; MULTIPLY BY 10 AND ADD
        STA 1,.EC11        ; SAVE SUM
        JMP .EC97          ; GET NEXT CHARACTER

.EC95:   LDA 1,.EC11        ; RESULT TO AC1
        MOVZL 1,1
        DSZ .EC10          ; TEST SIGN
        MOVZR 1,1,SKP      ; POSITIVE
        NEGOR 1,1          ; NEGATIVE
        LDA 2,.EC02        ; RESTORE AC2
        JMP @.EC03

; ROUTINE TO MULTIPLY AC1 BY 10 AND ADD AC0

.EC50:   MOVZL 1,2          ; N*2
        MOVZL 2,2          ; N*4
        ADD 2,1            ; N*5
        MOVZL 1,1          ; N*5*2 = N*10
        ADD 0,1            ; ADD AC0
        JMP 0,3            ; SUCCESS RETURN

.EC02:   0                 ; SAVE AC2
.EC03:   0                 ; SAVE AC3

.EC10:   0                 ; FLAG WORD FOR SIGN OF RESULT
.EC11:   0                 ; RUNNING SUM WORD

.EC20:   "+"              ; ASCII "+"
.EC21:   "-"              ; ASCII "-"
.EC22:   "0"              ; ASCII "0"
.EC23:   "9"              ; ASCII "9"
.EC24:   "S"              ; ASCII "S" FOR INDICATION
                           ; ENTRY

.EC40=40 ; ADDRESS OF GET CHARACTER
         ; ROUTINE
.EC41=41 ; ADDRESS OF PUT CHARACTER
         ; ROUTINE

```

NAME: BIND.SR.

PART NUMBER: 090-000030

DESCRIPTION: BINARY TO DECIMAL S.P.

REVISION HISTORY:

REV.	DATE
00	6/6/69
01	11/16/73

COPYRIGHT (C) DATA GENERAL CORPORATION, 1969, 1973
ALL RIGHTS RESERVED.

LICENSED MATERIAL - PROPERTY OF DATA GENERAL
CORPORATION

```
;
; BINARY TO DECIMAL ASCII CONVERT
; CONVERTS A SINGLE PRECISION, TWO'S COMPLEMENT
;   NUMBER TO AN ASCII CHARACTER STRING
;
; INPUT:          N IN AC1
;
; OUTPUT:         ASCII CHARACTER STRING, TERMINATED BY A
;                 NULL WORD
;                 CHARACTERS ARE RIGHT ADJUSTED IN AC0
;                 PASSED TO THE ROUTINE WHOSE ADDRESS
;                 MUST BE IN LOCATION 41 OF PAGE 0
;                 STRING OF FORM:
;                   +DDDDDD(NULL)
;                 OR
;                   -DDDDDD(NULL)
;
; CALLING SEQUENCE:
;   JSR          .BIND
;   RETURN
;
; DESTROYED:      AC1, AC3, CARRY
; UNCHANGED:      AC0, AC2
```

```
.BIND: STA 3, .ED03      ; SAVE RETURN
        STA 2, .ED02      ; SAVE AC2
        STA 0, .ED00      ; SAVE AC0
        LDA 3, .ED30      ; ADDRESS OF POWER OF TEN TABLE
        STA 3, .ED10      ; INITIALIZE POINTER
        LDA 0, .ED20      ; ASSUME NEGATIVE
        MOV# 1, 1, SZC
        NEG 1, 1, SKP
        LDA 0, .ED21      ; NO, IT IS POSITIVE; GET PLUS
```


.ED97:	STA 1,.ED11	; SAVE N
	JSR @.ED40	; PUT OUT SIGN OR DIGIT
	LDA 1,.ED11	; GET CURRENT VALUE OF N
	LDA 3,@.ED10	; GET CURRENT POWER OF TEN
	ISZ .ED10	; BUMP POINTER
	MOV 3,0,SNR	
	JMP .ED98	; PUT OUT NULL
	LDA 0,.ED22	; GET ASCII "0"
.ED99:	SUBZ 3,1,SZC	; DOES POWER OF TEN GO IN?
	INC 0,0,SKP	; YES, BUMP RESULT DIGIT
	ADD 3,1,SKP	; NO, RESTORE PREVIOUS VALUE
	JMP .ED99	; CONTINUE SUBTRACTING
	JMP .ED97	; PUT OUT DIGIT
.ED98:	JSR @.ED40	; PUT OUT NULL WORD
	LDA 0,.ED00	; RESTORE AC0
	LDA 2,.ED02	; RESTORE AC2
	JMP @.ED03	; RETURN
.ED00:	0	; SAVE AC0
.ED02:	0	; SAVE AC2
.ED03:	0	; SAVE AC3
	.RDX 10	
.ED05:	10000	; POWER OF TEN TABLE
	1000	; 10**3
	100	; 10**2
	10	; 10**1
	;	; 10**0
	0	; END OF TABLE INDICATION
	.RDX 8	
.ED10:	0	; ADDRESS OF CURRENT POWER OF ; TEN ENTRY
.ED11:	0	; RUNNING SUM WORD
.ED20:	"_"	; ASCII "_"
.ED21:	"+"	; ASCII "+"
.ED22:	60	; ASCII "0"
.ED30:	.ED05	; ADDRESS OF POWER OF TEN TABLE
.ED40=41		; PAGE ZERO PUT CHARACTER ; ROUTINE ADDRESS


```

*****
; NAME: SMPY.SR                                PART NUMBER: 090-000013
;
; DESCRIPTION: SIGNED MULTIPLY S.P.
;
; REVISION HISTORY:
;
;     REV.                DATE
;
;     00                  5/6/69
;     01                  11/16/73
;
; COPYRIGHT (C) DATA GENERAL CORPORATION, 1969, 1973
; ALL RIGHTS RESERVED.
;
; LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
; *****
; MULTIPLY
; MULTIPLIES TWO FIXED POINT, SINGLE PRECISION,
;     TWO'S COMPLEMENT NUMBERS
;
; INPUT:      N1 IN AC1, N2 IN AC2
;
; OUTPUT:     N1*N2, HIGH ORDER IN AC0, LOW ORDER IN
;             AC1
;
; CALLING SEQUENCE:
;             JSR .MPY
;             RETURN
;
; UNCHANGED:  AC2
; DESTROYED:  AC0, AC1, AC3, CARRY
;
; REQUIRES:   .MPYU (UNSIGNED MULTIPLY)

.MPY:  STA 3,.AB03          ; SAVE RETURN
      SUB 3,3              ; AC3 WILL CONTAIN CORRECTION
                          ; FACTOR
      MOVL # 1,1,SZC       ; REST SIGN OF N1
      ADD 2,3              ; ADD N2 TO CORRECTION
      MOVL# 2,2,SZC       ; TEST SIGN OF N2
      ADD 1,3              ; ADD N1 TO CORRECTION
      STA 3,.AB10          ; SAVE CORRECTION FACTOR
      JSR @.AB30           ; CALL UNSIGNED MULTIPLY
      LDA 3,.AB10          ; GET CORRECTION
      SUB 3,0              ; TRUE RESULT IN 2'S COMPLEMENT
                          ; FORM
      JMP @.AB03          ; RETURN

.AB03:  0                  ; SAVE AC3
.AB10:  0                  ; STORAGE FOR CORRECTION FACTOR
.AB30:  .MPYU              ; UNSIGNED MULTIPLY ADDRESS

```



```

; *****
;
; NAME: SDVD.SR                                PART NUMBER: 090-000014
;
; DESCRIPTION: SIGNED DIVIDE S. P.
;
; REVISION HISTORY:
;
;     REV.                DATE
;
;     00                  5/6/69
;     01                  11/16/73
;
; COPYRIGHT (C) DATA GENERAL CORPORATION, 1969, 1973
; ALL RIGHTS RESERVED.
; LICENSED MATERIAL - PROPERTY OF DATA GENERAL
; CORPORATION
; *****
;
; SIGNED DIVIDE
; DIVIDES TWO FIXED POINT, TWO'S COMPLEMENT NUMBERS
;
; INPUT:          N1 in AC0 and AC1 (HIGH AND LOW), N2 IN
;                  AC2
;
; OUTPUT:         N1/N2 ; REMAINDER IN AC0 (SAME SIGN AS
;                      DIVIDEND)
;                      QUOTIENT IN AC1
;
; CALLING SEQUENCE
;     JSR      .DIV
;     RETURN
;
; DESTROYED:      AC0, AC1, AC3, CARRY
; UNCHANGED:      AC2
;
; EXCEPTIONAL CONDITION:      IF THE MAGNITUDE OF THE
;                               QUOTIENT EXCEEDS
;                               @**15-1, CARRY IS SET AND
;                               THE DIVIDEND REMAINS UNCHANGED
;                               OTHERWISE, CARRY WILL BE 0
;
; REQUIRES:       .DIVU (UNSIGNED DIVIDE)
;
; .DIV:   STA 3, .AC03      ; SAVE RETURN
;         STA 2, .AC02      ; SAVE DIVISOR
;         STA 1, .AC01      ; SAVE DIVIDEND
;         STA 0, .AC00
;         MOVL 2, 3, SZC     ; CHECK SIGN OF DIVISOR
;         NEG 2, 2          ; FORM ABSOLUTE VALUE

```


SUBCL 3, 3	: SAVE SIGN OF DIVISOR IN AC3
MOVZL 3, 3	: POSITION IN BIT 14
MOVL# 0, 0, SNC	: TEST SIGN OF DIVIDEND
JMP .AC99	: POSITIVE, BIT 15 of AC3
	: CONTAINS SIGN OF DIVIDEND
INC 3, 3	: SIGN OF DIVIDEND TO
	: AC3 BIT 15
NEG 1, 1, SZR	: FORM ABS. VALUE OF DIVIDEND
COM 0, 0, SKP	
NEG 0, 0	
.AC99: STA 3, .AC10	: FLAG WORD FOR SIGNS OF
	: REMAINDER AND QUOTIENT
JSR @.AC30	: CALL .DIVI (UNSIGNED DIVIDE)
LDA 2, .AC02	: RESTORE AC2
MOVL 1, 1, SNC	: IF SIGN BIT SET,
	: QUOTIENT CAN'T
	: BE REPRESENTED IN 16 BITS
MOVR 1, 1, SZC	: IF CARRY SET, .DIVU GAVE
	: ERROR RETURN
JMP .AC98	: ERROR, RETURN WITH CARRY SET
LDA 3, .AC10	: GET FLAG WORD

; AC3 CONTAINS FOUR POSSIBLE COMBINATIONS

; THESE ARE: 00 Q POSITIVE, R POSITIVE
; 01 Q NEGATIVE, R NEGATIVE
; 10 Q NEGATIVE, R POSITIVE
; 11 Q POSITIVE, R NEGATIVE

MOVR3, 3, SNC	: TEST REMAINDER SIGN
COM 3, 3, SKP	: POSITIVE
NEG 0, 0	: REMAINDER IS NEGATIVE
MOVR 3, 3, SNC	: TEST QUOTIENT SIGN
NEG 1, 1	: QUOTIENT IS NEGATIVE
MOVZ 3, 3	: CLEAR CARRY
JMP @.AC03	: RETURN

.AC98: LDA 0, .AC00	: DIVIDE ERROR
LDA 1, .AC01	: RESTORE DIVIDEND
JMP @.AC03	: RETURN, CARRY IS SET

.AC00: 0	: SAVE DIVIDEND
.AC01: 0	: SAVE DIVIDEND
.AC02: 0	: SAVE AC2
.AC03: 0	: SAVE AC3

.AC10: 0	: SIGN OF QUOTIENT AND
	: REMAINDER FLAG WORD

.AC30: .DIVU	: UNSIGNED INTEGER DIVIDE
	: ROUTINE ADDRESS

APPENDIX G

INSTRUCTION EXECUTION TIMES

When two numbers are given, the one at the left of the slash is the time for an isolated transfer, the one at the right is the minimum time between consecutive transfers. All times are in usecs.

LDA	1.6
STA	1.6
ISZ, DSZ	1.8
JMP	.8
JSR	.8
Indirect addressing add	.8
Base register addressing add	0
Autoindexing add	.2
COM, NEG, MOV, INC	.8*
ADC, SUB, ADD, AND	.8*
*If skip occurs add	.2
IO input (except INTA)	2.2#
NIO	2.2#
IO output	2.2#
#S, C or P add	.6
IO skips	1.4*
INTA	2.2

MUL	8.8
DIV	8.8
Unsuccessful	1.6
Interrupt	1.6
With multiply-divide	10.6
Without multiply-divide	4.6
Data Channel	
Input	2.0
Output	2.0
Increment	2.2
Latency	3.6
High speed channel	
Input	.8
Output	.8/1.0
Increment	1.0/1.2
Latency	
With IO	3.6
Without IO	2.0

BIBLIOGRAPHY

- [A-1] Data Translation Inc., Engineering Specification Number 1600-674 Revision 1, DATA X Users Instruction Guide, 1974.
- [C-1] Cadzow, J. A. and Martens, H.R., Discrete-Time and Computer Control Systems, Prentice-Hall, Inc., 1970.
- [D-1] How to Use Nova Computers, Data General Corporation, October, 1972.
- [D-2] Data General Corporation, Program Tape 091-000036, Selfloading Bootstrap and Binary Loader, February, 1973.
- [D-3] Data General Corporation, Report 093-000062-03, The Stand-Alone Operating System User's Manual, June, 1973.
- [D-4] Data General Corporation, Program Tape 089-000080, Nova Text Editor, 1969.
- [D-5] Data General Corporation, Program Tape 091-00002, Program Assembler, 1969.
- [D-6] Data General Corporation, Program Tape 091-000017, Extended Assembler, 1969.
- [D-7] Data General Corporation, Program Tape 091-000016, Relocatable Loader, 1969.
- [D-8] Data General Corporation, Report 093-000067-01, Introduction to Programming Nova Computers, 1972.

12 APR 76
26 APR 76
13 JUN 76

23519
RENEWED
23554

Thesis
P7529 Pounds
c.1

102012

The Data General Nova
800 minicomputer as a
digital controller.

12 APR 76
26 APR 76
13 JUN 76

23519
RENEWED
23554

Thesis
P7529 Pounds
c.1

102012

The Data General Nova
800 minicomputer as a
digital controller.

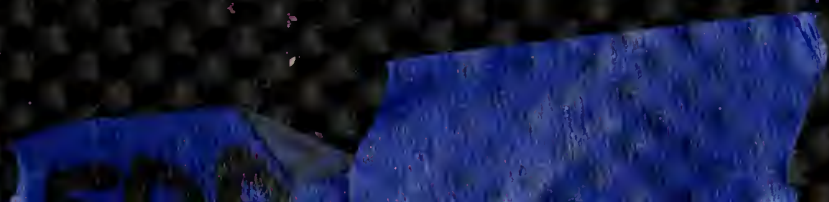
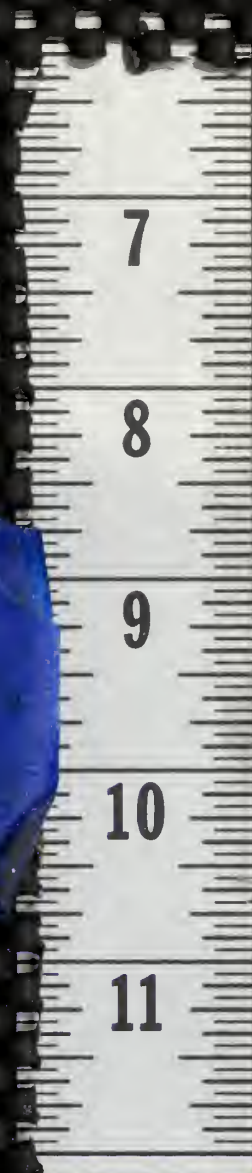
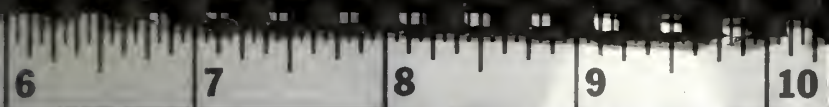
thesP7529

The Data General Nova 800 minicomputer a



3 2768 000 99303 4

DUDLEY KNOX LIBRARY

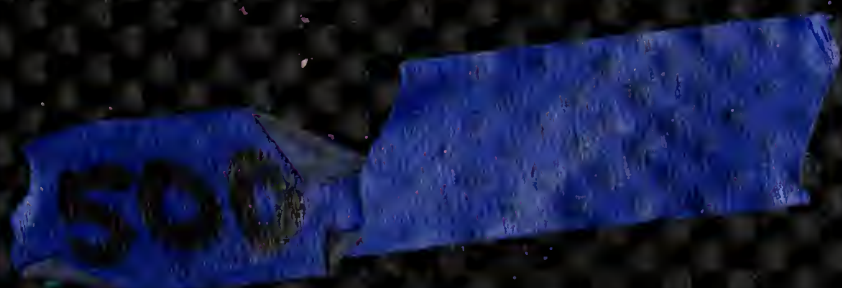
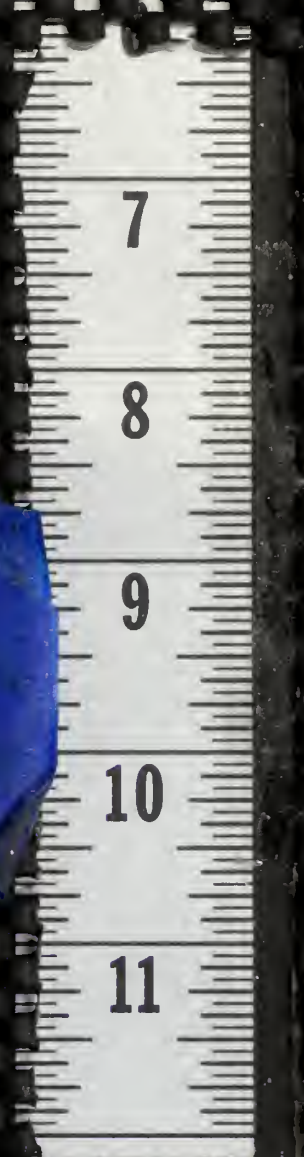
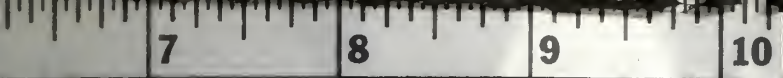


thesP7529

The Data General Nova 800 minicomputer a



3 2768 000 99303 4
DUDLEY KNOX LIBRARY

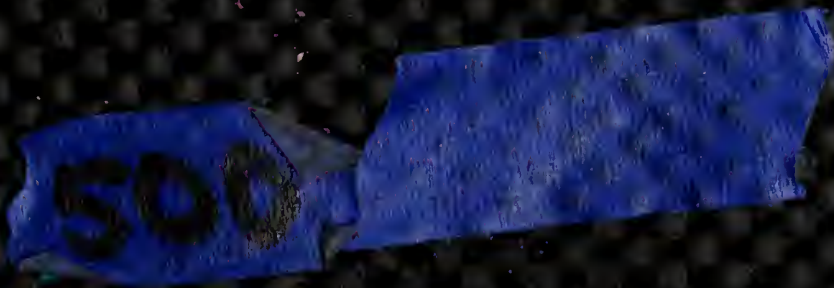
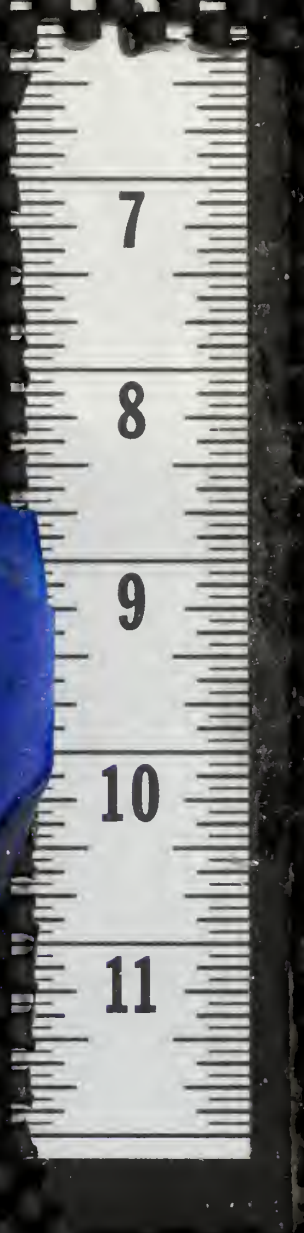
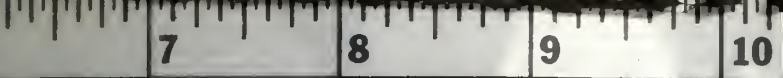


thesP7529

The Data General Nova 800 minicomputer a



3 2768 000 99303 4
DUDLEY KNOX LIBRARY



thesP7529

The Data General Nova 800 minicomputer a



3 2768 000 99303 4
DUDLEY KNOX LIBRARY

